Exhibit 2

eventually (in five years or more). *See also* nonvolatile memory.

permanent swap file \pər`mə-nənt swäp´ fīl\ *n.* In Windows; a file composed of contiguous disk sectors used for virtual memory operations. *See also* swap file, virtual memory.

permanent virtual circuit \pər`mə-nənt vər`-chōō-əl sər´kət\ *n. See* PVC.

permission \pər-mish´ən\ *n.* In a networked or multiuser computer environment, the ability of a particular user to access a particular resource by means of his or her user account. Permissions are granted by the system administrator or other authorized person; these permissions are stored in the system (often in a file called a *permissions log*) and are checked when a user attempts to access a resource.

perpendicular recording \pər-pən-dik`yə-lər rə-kōr´dēng\ *n.* A method of increasing storage capacity on magnetic media by aligning the magnetic dipoles, whose orientation determines bit values, in a direction that is perpendicular to the recording surface. *Also called* vertical recording.

persistence \pər-si´stəns\ *n.* A characteristic of some light-emitting materials, such as the phosphors used in CRTs, that causes an image to be retained for a short while after being irradiated, as by an electron beam in a CRT. The decay in persistence is sometimes called *luminance decay.*

persistent data \pər-si`stənt dā´tə, dat´ə\ *n.* Data that is stored in a database or on tape so that it is retained by the computer between sessions.

persistent link \pər-si`stənt lēnk´\ *n. See* hot link (definition 1).

persistent storage \pər-si`stənt stōr´əj\ *n.* Memory that remains intact when the power to a device is turned off, such as ROM. *See also* memory.

personal computer \pər`sə-nəl kəm-pyōō´tər\ *n.* A computer designed for use by one person at a time. Personal computers do not need to share the processing, disk, and printer resources of another computer. IBM PC-compatible computers and Apple Macintoshes are both examples of personal computers. *Acronym:* PC (P-C´).

Personal Computer \pər`sə-nəl kəm-pyōō´tər\ *n. See* IBM PC.

Personal Computer Memory Card International Association \pər`sən-əl kəm-pyōō-tər mem`ər-ē kärd in-tər-nash`ə-nəl ə-sō`sē-ā´shən\ *n. See* PCMCIA.

personal digital assistant \pər`sə-nəl dij`i-təl ə-si´stənt\ *n. See* PDA.

personal finance manager \pər`sə-nəl fī´nans man`ə-jər\ *n.* A software application designed to assist the user in performing simple financial accounting tasks, such as balancing checkbooks and paying bills.

personal information manager \pər`sə-nəl in-fər-mā´shən man`ə-jər\ *n. See* PIM.

perspective view \pər-spek´tiv vyōō`\ *n.* In computer graphics, a display method that shows objects in three dimensions (height, width, and depth), with the depth aspect rendered according to the desired perspective. An advantage of perspective view is that it presents a more accurate representation of what the human eye perceives. *Compare* isometric view.

peta- \pet´ə\ *prefix* Abbreviated P. Denotes 1 quadrillion (10^{15}). In computing, which is based on the binary (base-2) numbering system, *peta-* has a literal value of 1,125,899,906,842,624, which is the power of 2 (2^{50}) closest to 1 quadrillion.

petabyte \pet´ə-bīt`\ *n.* Abbreviated PB. Either 1 quadrillion bytes or 1,125,899,906,842,624 bytes.

.pg \dot`P-G´\ *n.* On the Internet, the major geographic domain specifying that an address is located in Papua New Guinea.

PGA \P`G-A´\ *n. See* pin grid array, Professional Graphics Adapter.

PgDn key \pāj-doun´ kē`\ *n. See* Page Down key.

PGP \P`G-P´\ *n.* Acronym for **P**retty **G**ood **P**rivacy. A program for public key encryption, using the RSA algorithm, developed by Philip Zimmermann. PGP software is available in unsupported free versions and supported commercial versions. *See also* privacy, public key encryption, RSA encryption.

PgUp key \pāj-up´ kē`\ *n. See* Page Up key.

.ph \dot`P-H´\ *n.* On the Internet, the major geographic domain specifying that an address is located in the Philippines.

phase \fāz\ *n.* A relative measurement that describes the temporal relationship between two signals that have the same frequency. Phase is measured in degrees, with one full oscillation cycle having 360 degrees. The phase of one signal

process or input in I/O mode. This information occupies the top of the stack segment. See also user block.

persistence The length of time required for the light from a phosphor on a display screen to fade. See also phosphor.

persistent session In the NetView program, a network management session that remains active even though there is no activity on the session for a specified period of time.

personal code A code that identifies a user to a computer system. See also password.

personal computer (PC) (1) A microcomputer primarily intended for stand-alone use by an individual. (T) (2) A desk-top, floor-standing, or portable microcomputer that usually consists of a system unit, a display monitor, a keyboard, one or more diskette drives, internal fixed-disk storage, and an optional printer. PCs are designed primarily to give independent computing power to a single user and are inexpensively priced for purchase by individuals or small businesses. See also desktop computer, laptop computer, mainframe, microcomputer, minicomputer, portable computer, supermini.

personal document A document that is intended to be handled only by its owner or by someone who knows the personal document password specified by the owner.

perspective projection A graphical technique used to achieve realism when drawing primitives. In a perspective projection, the lines of projection meet at the viewpoint; thus, the size of a primitive varies inversely with its distance from the source projection. The farther a primitive or part of a primitive is from the viewer, the smaller it will be drawn. This effect, known as perspective foreshortening, is similar to the effect achieved by photography and by the human visual system. See also orthographic projection.

peta Two to the fiftieth power.

PF key Programmed function key.

PFM Program fault management.

PFT Page frame table.

PGB Presentation services global block.

PgDn See Page Down.

PGF Presentation graphics feature.

PG indicator See program-mode indicator.

PGR Presentation graphics routines.

PgUp See Page Up.

PH Phase. Also shown by the Greek letter omega (Ω).

phantom circuit A superimposed circuit derived from two side circuits, suitably arranged pairs of wires with each pair being a circuit that acts as one conductor of the phantom circuit.

phantom hyphen Synonym for soft hyphen. (T)

phase (1) A distinct part of a process in which related operations are performed. (2) A part of a sort/merge program; for example, sort phase, merge phase. (3) A part of a data call. See data transfer phase, network control phase. (4) In VSE, the smallest complete unit of executable code that can be loaded into virtual storage. (5) See assembly phase, compile phase, execute phase, translate phase.

Phase Alternation Line (PAL) The television broadcast standard for European video outside of France and the countries of the former Soviet Union.

phase coherent FSK Frequency shift keying where the two signaling frequencies are integrally related to the data rate and transitions between the signaling frequencies are made at zero crossings of the carrier waveform. (T)

phase continuous FSK Frequency shift keying where the transition between signaling frequencies is accomplished by a continuous change of frequency (as opposed to the discontinuous replacement of one frequency by another, such as might be accomplished by a switch); thus, it is also a form of frequency modulation. (T)

phase distortion See distortion.

phase encoding (1) Encoding in which the phase of the wave is used to encode digital data; for example, Manchester encoding. (T) (2) Synonym for phase modulation recording.

phase jitter A form of perturbation that causes intermittent variations of the phase of signals. (T)

phase modulation Modulation that varies the phase angle of a sinusoidal carrier from a reference carrier phase angle by an amount proportional to the instantaneous amplitude of the modulating signal.

phase modulation recording A magnetic recording in which each storage cell is divided into two regions that are magnetized in opposite senses. The sequence of these senses indicates whether the binary character

For each independent variable, a regression analysis can determine the correlation coefficient of the independent variable—that is, the degree to which variations in the independent variable cause changes in the dependent variable. *See also* dependent variable.

multiple-user system \mul`tə-pl-yo͞o`zər si´stəm\ *n.* *See* multiuser system.

multiplexer \mul´tə-pleks`ər\ *n.* **1.** A hardware circuit for selecting a single output from multiple inputs. **2.** A device for funneling several different streams of data over a common communications line. Multiplexers are used either to attach many communications lines to a smaller number of communications ports or to attach a large number of communications ports to a smaller number of communications lines. *Acronym:* MUX (M`U-X´, muks).

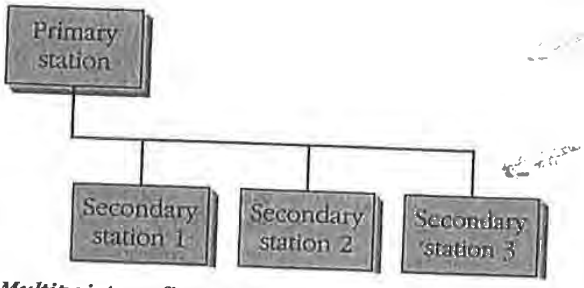
multiplexer channel \mul´tə-pleks`ər chan`əl\ *n.* One of the inputs to a multiplexer. *See also* multiplexer (definition 1).

multiplexing \mul´tə-pleks`ēng\ *n.* A technique used in communications and input/output operations for transmitting a number of separate signals simultaneously over a single channel or line. To maintain the integrity of each signal on the channel, multiplexing can separate the signals by time, space, or frequency. The device used to combine the signals is a *multiplexer. See also* FDM, space-division multiplexing, time-division multiplexing.

multiplicand \mul´tə-pli-kand`\ *n.* In arithmetic, the number that is multiplied by another number (the multiplier). In mathematics, the multiplicand and the multiplier are interchangeable, depending on how the problem is stated, because the result is the same if the two are reversed—for example, 2 × 3 and 3 × 2. In arithmetic performed by computers, however, the multiplicand is different from the multiplier because computer multiplication is usually performed as addition. Therefore, 2 × 3 means "add 2 three times," whereas 3 × 2 means "add 3 two times." *See also* factor. *Compare* multiplier (definition 1).

multiplier \mul´tə-plī`ər\ *n.* **1.** In arithmetic, the number that indicates how many times another number (the multiplicand) is multiplied. *See also* factor. *Compare* multiplicand. **2.** In computing, an electronic device independent of the central processing unit (CPU) that performs multiplication by adding the multiplicand according to the value of the digits in the multiplier.

multipoint configuration \mul`tī-point kən-fi-gyər-ā´shən, mul`tē-point\ *n.* A communications link in which multiple stations are connected sequentially to the same communications line. Typically, the communications line is controlled by a primary station, such as a computer, and the stations attached to the line are secondary. See the illustration.



Multipoint configuration.

multiprocessing \mul`tī-pros´es-ēng, mul`tē-pros´es-ēng\ *n.* A mode of operation in which two or more connected and roughly equal processing units each carry out one or more processes (programs or sets of instructions) in tandem. In multiprocessing, each processing unit works on a different set of instructions or on different parts of the same process. The objective is increased speed or computing power, the same as in parallel processing and in the use of special units called *coprocessors. Compare* coprocessor, parallel processing.

Multi-Protocol Over ATM \mul`tē-prō`tə-kol ōvər A`T-M´, mul`tī-prō`tə-kol\ *n.* *See* MPOA.

Multipurpose Internet Mail Extensions \mul`tē-pur-pəs in`tər-net māl´ eks-ten`shənz, mul`tī-pur´pəs\ *n.* *See* MIME.

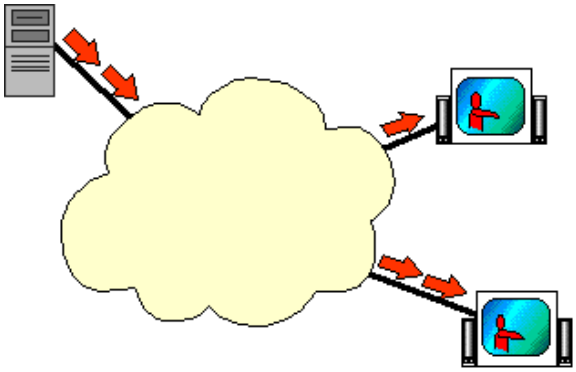
multiscan monitor \mul`tē-skan mon´ə-tər, mul`tī-skan\ *n.* A computer monitor capable of operating at different scanning frequencies to accommodate different screen resolutions.

multisync monitor \mul`tī-sēnk mon´ə-ter, mul`tē-sēnk\ *n.* A monitor capable of responding to a wide range of horizontal and vertical synchronization rates. Such a monitor can be used with a vari-

MPEG-2 Transmission

erg.abdn.ac.uk/future-net/digital-video/mpeg2-trans.html

Single Program Transport Stream (Audio and Video PES).



The MPEG-2 standards define how to format the various component parts of a multimedia programme (which may consist of: MPEG-2 compressed video, compressed audio, control data and/or user data). It also defines how these components are combined into a single synchronous transmission bit stream. The process of combining the steams is known as *multiplexing*.

The multiplexed stream may be transmitted over a variety of links, standards / products are (or will soon be) available for :

- Radio Frequency Links (UHF/VHF)
- Digital Broadcast Satellite Links
- Cable TV Networks
- Standard Terrestrial Communication Links (PDH, SDH)
- Microwave Line of Sight (LoS) Links (wireless)
- Digital Subscriber Links (ADSL family)
- Packet / Cell Links (ATM, IP, IPv6, Ethernet)

To understand how the component parts of the bit stream are multiplexed, we need to first look at each component part. The most basic component is known as an *Elementary Stream* in MPEG. A programme (perhaps most easily thought of as a television programme, or a Digital Versatile Disk (DVD) track) contains a combination of elementary streams (typically one for video, one or more for audio, control data, subtitles, etc).

Each *Elementary Stream (ES)* output by an MPEG audio, video and (some) data encoders contain a single type of (usually compressed) signal. There are various forms of ES, including:

- Digital Control Data
- Digital Audio (sampled and compressed)
- Digital Video (sampled and compressed)
- Digital Data (synchronous, or asynchronous)

For video and audio, the data is organised into *access units*, each representing a fundamental unit of encoding. For example, in video, an access unit will usually be a complete encoded video frame.

Each ES is input to an MPEG-2 processor (e.g. a video compressor or data formatted) which accumulates the data into a stream of *Packetised Elementary Stream (PES)* packets. A PES packet may be a fixed (or variable) sized block, with up to 65536 bytes per block and includes a 6 byte protocol header. A PES is usually organised to contain an integral number of ES access units.

The PES header starts with a 3 byte start code, followed by a one byte stream ID and a 2 byte length field.

- 110x xxxx - MPEG-2 audio stream number x xxxx.
- 1110 yyyy - MPEG-2 video stream number yyyy.
- 1111 0010 - MPEG-2 DSM-CC control packets.

The next field contain the *PES Indicators*. These provide additional information about the stream to assist the decoder at the receiver. The following indicators are defined:

- PES_Scrambling_Control - Defines whether scrambling is used, and the chosen scrambling method.
- PES_Priority - Indicates priority of the current PES packet.
- data_alignment_indicator - Indicates if the payload starts with a video or audio start code.
- copyright information - Indicates if the payload is copyright protected.
- original_or_copy - Indicates if this is the original ES.

A one byte flags field completes the PES header. This defines the following optional fields, which if present, are inserted before the start of the PES payload.

- *Presentation Time Stamp (PTS)* and possibly a *Decode Time Stamp (DTS)* - For audio / video streams these time stamps which may be used to synchronise a set of elementary streams and control the rate at which they are replayed by the receiver.
- *Elementary Stream Clock Reference (ESCR)*
- *Elementary Stream rate* - Rate at which the ES was encoded.
- *Trick Mode* - indicates the video/audio is not the normal ES, e.g. after DSM-CC has signalled a replay.
- Copyright Information - set to 1 to indicated a copyright ES.
- CRC - this may be used to monitor errors in the previous PES packet
- PES Extension Information - may be used to support MPEG-1 streams.

The PES packet payload includes the ES data. The information in the PES header is, in general, independent of the transmission method used.

- **MPEG Program Stream** A group of tightly coupled PES packets referenced to the same time base. Such streams are suited for transmission in a relatively error-free environment and enable easy software processing of the received data. This form of multiplexing is used for video playback and for some network applications.
- **MPEG Transport Stream** Each PES packet is broken into fixed-sized transport packets forming a general purpose way of combining one or more streams, possibly with independent time bases. This is suited for transmission in which there may be potential packet loss or corruption by noise, or / and where there is a need to send more than one programme at a time.

A transport stream consists of a sequence of fixed sized transport packet of 188 B. Each packet comprises 184 B of payload and a 4 B header. One of the items in this 4 B header is the 13 bit *Packet Identifier (PID)* which plays a key role in the operation of the Transport Stream.

The format of the transport stream is described using the figure below (a later section describes the detailed format of the TS packet header). This figure shows two elementary streams sent in the same MPEG-2 transport multiplex. Each packet is associated with a PES through the setting of the PID value in the packet header (the values of 64 and 51 in the figure). The audio packets have been assigned PID 64, and the video packets PID 51 (these are arbitrary, but different values). As is usual, there are more video than audio packets, but you may also note that the two types of packets are not evenly spaced in time. The MPEG-TS is not a time division multiplex, packets with any PID may be inserted into the TS at any time by the TS multiplexor. If no packets are available at the multiplexor, it inserts null packets (denoted by a PID value of 0x1FFF) to retain the specified TS bit rate. The multiplexor also does not synchronise the two PESs, indeed the encoding and decoding delay for each PES may (and usually is different). A separate process is therefore require to synchronise the two streams (see below).

Transmission of the MPEG-TS

Although the MPEG TS may be directly used over a wide variety of media (as in DVB), it may also be used over a communication network. It is designed to be robust with short frames, each one being protected by a strong error correction mechanism. It is constructed to match the characteristics of the generic radio or cable channel and expects an uncorrected *Bit Error Rate* (BER) of better than 10^{-10}. (The different variants of DVB each have their own outer coding and modulation methods designed for the particular environment.)

The MPEG-2 Transport Stream is so called, to signify that it is the input to the *Transport Layer* in the ISO *Open System Interconnection (OSI)* seven-layer network reference model. It is not, in itself, a transport layer protocol and no mechanism is provided to ensure the reliable delivery of the transported data. MPEG-2 relies on underlying layers for such

services. MPEG-2 transport relies on underlying layers for such services. MPEG-2 transport requires the underlying layer to identify the transport packets, and to indicate in the transport packet header, when a transport packet has been erroneously transmitted.

When the MPEG-TS is used over a lower layer network protocol, the lower layer must identify the start of each transport packets, and indicate in the transport packet header, when a transport packet has been erroneously received. The MPEG TS packet size also corresponds to eight *Asynchronous Transfer Mode (ATM)* cells, assuming 8 B overhead (associated with the *ATM Adaptation Layer (AAL)*).

Most transport streams consist of a number of related elementary streams (e.g. the video and audio of a TV programme). The decoding of the elementary streams may need to be co-ordinated (synchronised) to ensure that the audio playback is in synchronism with the corresponding video frames. Each stream may be tightly synchronised (usually necessary for digital TV programs, or for digital radio programs), or not synchronised (in the case of programs offering downloading of software or games, as an example). To help synchronisation time stamps may be (optionally) sent in the transport stream.

They are two types of time stamps:

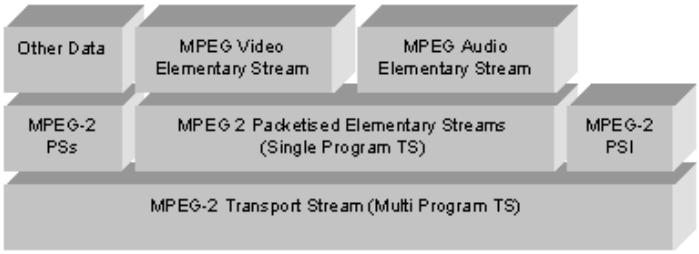
- The first type is usually called a *reference time stamp*. This time stamp is the indication of the current time. Reference time stamps are to be found in the PES syntax (ESCR), in the program syntax (SCR), and in the transport packet adaption *Program Clock Reference (PCR)* field.
- The second type of time stamp is called *Decoding Time Stamp (DTS*) or *Presentation Time Stamp (PTS)*. These time stamps are inserted close to the material to which they refer (normally in the PES packet header). They indicate the exact moment where a video frame or an audio frame has to be decoded or presented to the user respectively. These rely on reference time stamps for operation.

Single and Multiple Program Transport Streams

A TS may correspond to a single TV programme, or multimedia stream (e.g. with two a video PES and an audio PES). This type of TS is normally called a *Single Programme Transport Stream (SPTS)*.

An SPTS contains all the information requires to reproduce the encoded TV channel or multimedia stream. It may contain only an audio and video PESs, but in practice there will be other types of PES as well. Each PES shares a common timebase. Although some equipments output and use SPTS, this is not the normal form transmitted over a DVB link.

In most cases one or more SPTS streams are combined to form a *Multiple Programme Transport Stream (MPTS)*. This larger aggregate also contains all the control information *(Program Specific Information (PSI))* required to co-ordinate the DVB system, and any other data which is to be sent.



Streams supported by the MPTS

For a user to receive a particular transport stream, the user must first determine the PID being used, and then filter packets which have a matching PID value. To help the user identify which PID corresponds to which programme, a special set of streams, known as *Signalling Tables*, are transmitted with a description of each program carried within the MPEG-2 Transport Stream. Signalling tables are sent separately to PES, and are not synchronised with the elementary streams (i.e they are an independent control channel).

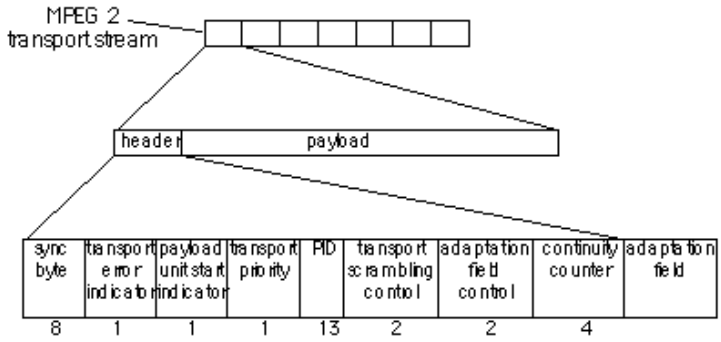
The tables (called *Program Specific Information (PSI)* in MPEG-2) consist of a description of the elementary streams which need to be combined to build programmes, and a description of the programmes. Each PSI table is carried in a sequence of PSI *Sections*, which may be of variable length (but are usually small, c.f. PES packets). Each section is protected by a *CRC* (checksum) to verify the integrity of the table being carried. The length of a section allows a decoder to identify the next section in a packet. A PSI section may also be used for down-loading data to a remote site. Tables are sent periodically by including them in the transmitted transport multiplex.

To identify the required PID to de-multiplex a particular PES, the user searches for a description in a particular table, the *Program Association Table (PAT)*. This lists all programmes in the multiplex. Each programme is associated with a set of PIDs (one for each PES) which correspond to a *Programme Map Table (PMT)* carried as a separate PSI section. There is one PMT per programme. DVB also adds a number of additional tables including those shown below.

In addition to the PSI carried in each multiplex (MPTS), a service also carries information relating to the service as a whole. Since a service may use a number of MPTS to send all the required programs. Information is provided in the PSI tables defined by DVB. Each PSI table refers to the MPTS in which it is carried and any other MPTSs which carry other TS which are offered as a part of the same service.

Most viewers have little knowledge of the operation of these tables and interact with the decoder through a graphical or textual programme guide.

Each MPEG-2 TS packet carries 184 B of payload data prefixed by a 4 B (32 bit) header.

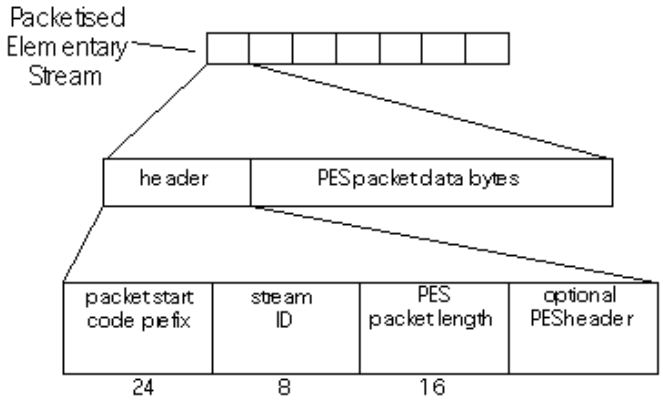


- The header starts with a well-known *Synchronisation Byte (8 bits)*. This has the bit pattern 0x47 (0100 0111).
- A set of three flag bits are used to indicate how the payload should be processed.
  1. The first flag indicates a transport error.
  2. The second flag indicates the start of a payload (payload_unit_start_indicator)
  3. The third flag indicates transport priority bit.

- The flags are followed by a 13 bit *Packet Identifier (PID)*.This is used to uniquely identify the stream to which the packet belongs (e.g. PES packets corresponding to an ES) generated by the multiplexer. The PID allows the receiver to differentiate the stream to which each received packet belongs. Some PID values are predefined and are used to indicate various streams of control information. A packet with an unknown PID, or one with a PID which is not required by the receiver, is silently discarded. The particular PID value of 0x1FFF is reserved to indicate that the packet is a null packet (and is to be ignored by the receiver).
- The two scrambling control bits are used by conditional access procedures to encrypted the payload of some TS packets.
- Two adaption field control bits which may take four values:
  1. 01 – no adaptation field, payload only
  2. 10 – adaptation field only, no payload
  3. 11 – adaptation field followed by payload
  4. 00 - RESERVED for future use

- Finally there is a half byte *Continuity Counter* (4 bits)
- The simplest option, from both the encoder and receiver viewpoints, is to send only one PES (or a part of single PES) in a TS packet. This allows the TS packet header to indicate the start of the PES, but since a PES packet may have an arbitrary length, also requires the remainder of the TS packet to be padded, ensuring correct alignment of the next PES to the start of a TS packet. In MPEG-2 the padding value is the hexadecimal byte 0xFF.

- In general a given PES packet spans several TS packets so that the majority of TS packets contain continuation data in their payloads. When a PES packet is starting, however, the payload_unit_start_indicator bit is set to '1' which means the first byte of the TS payload contains the first byte of the PES packet header. Only one PES packet can start in any single TS packet. The TS header also contains the PID so that the receiver can accept or reject PES packets at a high level without burdening the receiver with too much processing. This has an impact on short PES packets



MPEG PES mapping onto the MPEG-2 TS

The presence of an adaptation field is indicated by the adaption field control bits in a transport stream packet. If present, the adaption field directly follows the 4 B packet header, before any user payload data. It may contain a variety of data used for timing and control.

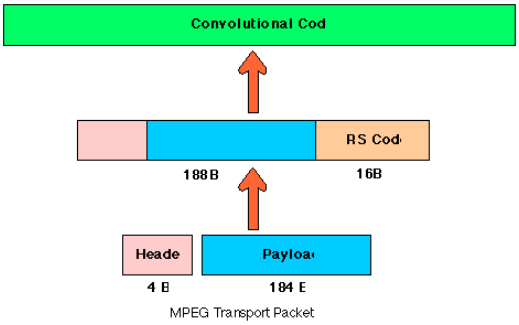
One important item in most adaption packets is the *Program Clock Reference (PCR)* field.

Another important item is *splice_countdown* field. This field is used to indicate the end of a series of ES access units. It allows the MPEG-2 TS multiplexor to determine appropriate places in a stream were the video may be spliced to another video source without introducing undesirable disruption to the video replayed by the receiver. Since MPEG-2 video uses inter-frame coding a seamless switch-over between sources can only occur on an I-frame boundary (indicated by a splice count of 0). This feature may, for instance be used to insert a news flash in a scheduled TV transmission.

One other bit of interest here is the *transport_private_data_flag* which is set to 1 when the adaptation field contains private data bytes. Another is the *transport_private_data_length field* which specifies how many private data bytes will follow the field. Private data is not allowed to increase the adaptation field beyond the TS payload size of 184 bytes.

DVB transmission via satellite (often known as DVB-S), defines a series of options for sending MPEG-TS packets over satellite links. The DVB-S standard requires the 188 B (scrambled) transport packets to be protected by 16 bytes of Reed Solomon (RS) coding.

MPEG Transport Service Encoding Specified by DVB-S



The resultant bit stream is then interleaved and convolutional coding is applied. The level of coding may be selected by the service provider (from 1/2 to 7/8 depending on the intended application and available bandwidth). The digital bit stream is then modulated using *Quadrature Phase Shift Keying (QPSK)*. A typical satellite channel has a 36 MHz bandwidth, which may support transmission at up to 35-40 Mbps (assuming delivery to a 0.5m receiving antenna).

DSM-CC is a toolkit for developing control channels associated with MPEG-1 and MPEG-2 streams. It uses a client/server model connected via an underlying network (carried via the MPEG-2 multiplex or independently if needed). DSM-CC may be used for controlling the video reception, providing features normally found on *Video Cassette Recorders* (VCR) (fast-forward, rewind, pause, etc). It may also be used for a wide variety of other purposes including packet data

♦ The Session Initiation Protocol: Providing Advanced Telephony Services Across the Internet

Henning G. Schulzrinne and Jonathan D. Rosenberg

During the past few years, Internet telephony has evolved from a toy for the technically savvy to a technology that, in the not too distant future, may replace the existing circuit-switched telephone network. Supporting the widespread use of Internet telephony requires a host of standardized protocols to ensure quality of service (QoS), transport audio and video data, provide directory services, and enable signaling. Signaling protocols are of particular interest because they are the basis for advanced services such as mobility, universal numbers, multiparty conferencing, voice mail, and automatic call distribution. Two signaling protocols have emerged to fill this need: the ITU-T H.323 suite of protocols and session initiation protocol (SIP), developed by the Internet Engineering Task Force (IETF). In this paper we examine how SIP is used in Internet telephony. We present an overview of the protocol and its architecture, and describe how it can be used to provide a number of advanced services. Our discussion of some of SIP's strengths—its simplicity, scalability, extensibility, and modularity—also analyzes why these are critical components for an IP telephony signaling protocol. SIP will prove to be a valuable tool, not just for end-to-end IP services, but also for controlling existing phone services.

Introduction

The term *Internet telephony* has evolved to infer a range of different services. In general, it refers to the transport of real-time media—such as voice and video—over the Internet to provide interactive communications among Internet users. The parties involved may access the Internet via a PC, a stand-alone Internet protocol (IP)-enabled device, or even by dialing up to a gateway from the handset of a traditional public switched telephone network (PSTN).

The advantages of IP telephony are quite sweeping.[1] It offers high-quality voice, improved multiplexing gains (an inherent advantage in any packet-switched network), rich computer telephony integration, advanced services, an open market for providers, and reduced cost.

Supporting Internet telephony requires a variety of components, each of which can be supported by several protocols:

- Transport, provided by the real-time transport protocol (RTP);[2]
- QoS, provided by the resource reservation protocol (RSVP),[3,4] yet another sender session Internet reservations (YESSIR)—both of which make end-to-end resource reservations on the Internet[5]—or differentiated services;[6]
- Authentication, authorization, and accounting, provided by the remote authentication dial-in user service (RADIUS)[7,8] and DIAMETER;[9]
- Gateway discovery,[10] provided by the gateway location protocol;[11] and

Panel 1. Abbreviations, Acronyms, and Terms

ACD—automatic call distribution
BGP—border gateway protocol
CGI—common gateway interface
DIAMETER—a protocol for authentication, authorization, and accounting
DNS—Domain Name System
GSTN—global switched telephone network
HTML—HyperText Markup Language
HTTP—hypertext transport protocol
IANA—Internet Assigned Numbers Authority
IETF—Internet Engineering Task Force
IN—intelligent network
IP—Internet protocol
IPtel—Internet protocol telephony
ISDN—integrated services digital network
ISP—Internet service provider
ITU–T—International Telecommunication Union–Telecommunication Standardization Sector
JTAPI—Java telephony application programming interface
LDAP—lightweight directory access protocol
Mbone—multicast backbone
MIME—multipurpose Internet mail extension
MTA—message transfer agent
MX—mail exchange
PBX—private branch exchange
PEP—protocol extensions protocol
Perl—Practical Extraction Report Language, a text processing language
PINT—PSTN and IP Internetworking
POTS—"plain old telephone service"
PSTN—public switched telephone network
QoS—quality of service
RADIUS—remote authentication dial-in user service
RSVP—resource reservation protocol
RTP—real-time transport protocol
RTSP—real-time streaming protocol
SCP—service control point
SDP—session description protocol
sdr—an mbone session directory tool
SIP—session initiation protocol
SMIL—Synchronized Multimedia Integration Language
SMTP—simple mail transfer protocol
SRV—a DNS resource record for servers
TCP—transmission control protocol
UAC—user agent client
UAS—user agent server
UDP—user datagram protocol
URI—uniform resource identifier
URL—uniform resource location
VPN—virtual private network
XML—Extensible Markup Language
YESSIR—yet another sender session Internet reservations

- Directory service, provided by the lightweight directory access protocol (LDAP).[12]

In addition, Internet telephony requires a means for prospective communications partners to find each other and to signal to one another their desire to communicate. We refer to this functionality as *Internet telephony signaling*.

Internet telephony signaling encompasses a number of functions, including:

- *Name translation and user location*, which involve the mapping between names to identify a callee and the eventual location of the callee, be it a telephone number, pager, voice mail, e-mail address, or Web page. This translation and location function can be more complex than a simple database lookup, but it may depend on caller and callee preferences; media and codec support at various points of presence; and service provider, or third-party, logic.
- *Feature negotiation*, which allows a group of participants to agree on the media to exchange, and their respective parameters. In a multiparty IP telephony conference, the set and type of media need not be uniform. Different participants can exchange different media types with each other; some may only receive audio, while others may receive audio and video. Such a mix may be due to limited computing facilities at an endpoint, a desire to use a particular format for a medium, or the lack of a single medium format common to all participants.
- *Call participant management*, which allows any call participant to invite new users into an existing call and terminate associations with other participants. During the call, participants

should be able to transfer other participants and place them on hold.

- *Call feature changes*, which make it possible to adjust the composition of media sessions during the course of a call, either because the participants require additional or reduced functionality, or because of constraints imposed or removed by adding or removing call participants.

Two protocols have emerged to provide these functions, the International Telecommunication Union—Telecommunication Standardization Sector (ITU-T) H.323 series of recommendations,[13] and the Internet Engineering Task Force (IETF) session initiation protocol (SIP).[14] This paper presents SIP and describes how it is used for various advanced services. In an earlier work we compared these services with H.323.[15] The remainder of this paper is organized as follows: First, we present an overview of SIP and its basic functionality, followed by an analysis of several network services enabled by SIP. Next we describe how SIP supports client-to-client services, and then we review its benefits.

Overview of SIP

SIP—used to establish, change, and tear down calls between one or more endpoints in an IP-based network—is based heavily on some of the most successful protocols to emerge from the IETF, which standardizes protocols used on the Internet. In particular, SIP is modeled after the simple mail transfer protocol (SMTP),[16,17] the basis for e-mail, and the hypertext transfer protocol (HTTP),[18] the basis of the Web. Like both of these, SIP is a textual client-server protocol, in which requests are issued by the client and responses are returned by the server.

SIP reuses much of the syntax and semantics of HTTP, including its response code architecture, many message headers, and its overall operation. SIP maps each IP telephony function to one or more transaction requests issued by a client, and one or more responses returned by one or more servers. Like HTTP transactions, SIP transactions are idempotent. Also like HTTP, each SIP request is an attempt to invoke some method on the server. Of the six SIP methods that exist, the most basic is the `INVITE` method, used to initiate a call between the client and the server.

Unlike HTTP and SMTP, SIP can run on top of either the transmission control protocol (TCP) or the user datagram protocol (UDP). SIP provides its own mechanisms for reliability, and UDP enables SIP messages to be multicast. Multicasting allows for, among other features, group invitations and basic automatic call distribution (ACD) functions that do not require a distribution server. By avoiding the TCP synchronization handshake, UDP facilitates fast operation; removing the need for the TCP state in the kernel provides better scalability. When used with TCP, SIP allows many requests and responses to be sent over the same TCP connection, as in HTTP 1.1.[18]

Protocol Components

A SIP system has only two components: user agents and network servers. A *user agent* is an end system that acts on behalf of someone who wants to participate in calls. In general, a user agent contains both a protocol client—called a user agent client (UAC)—and a protocol server—called a user agent server (UAS). The UAC is used to initiate a call, and the UAS is used to answer a call. The presence of both in a user agent enables peer-to-peer operation to take place using a client-server protocol.

In addition to user agents, SIP provides for two different types of network servers: proxy and redirect. A SIP proxy acts in much the same way as an HTTP proxy or an SMTP message transfer agent (MTA). It receives a request, determines which server to send it to, and then forwards the request, possibly after modifying some of the header fields. A SIP proxy has no way of knowing whether the next server to receive the request is another proxy server, a redirect server, or a UAS. For this reason, SIP requests can traverse many servers on their way from UAC to UAS. Responses to a request always travel along the same set of servers the request followed, but in reverse order.

A redirect server receives requests, but instead of forwarding them to the next hop server, it tells the client to contact the next hop server directly. It answers the client's request using a redirect response,

which contains the address of the next hop server. This procedure is analogous to iterative searches in the Domain Name System (DNS),[19,20] just as proxying is analogous to recursive searches.

SIP Network Servers

The main function of a SIP network server is to provide for name resolution and user location. When a user wants to place a call, the SIP UAC sends an `INVITE` request. In general, the caller will not know the IP address or host name of the UAS for the given user; it will only have a name—usually an e-mail address, but sometimes a telephone number or another local identifier—that represents the caller. Using this name, the UAC can determine which network server may be able to resolve the name to an IP address. This network server may, in turn, proxy or redirect the call to additional servers, eventually arriving at one that definitively knows the IP address where the user can be contacted. The process of determining the next-hop server is known as *next-hop routing*. Similar to other dynamic routing protocols, like the border gateway protocol (BGP),[21] SIP provides facilities for loop detection and prevention.

To determine the next-hop server, a SIP network server can use any means at its disposal, such as searching the DNS, accessing databases, executing programs, or prompting users. The final UAS contacted by the caller is determined by the composition of the decisions made at all the servers from caller to callee. The ability of SIP servers to route calls based on any means at their disposal makes them the basis for powerful mobility and forwarding services.

As a result of its next-hop routing decision, a SIP network server may determine that several next-hop servers may be able to contact the user. In these cases, SIP allows a proxy server to *fork* an incoming request, sending it in parallel to multiple next-hop servers. Under normal conditions, each server will generate a response; SIP has rules for merging and returning these responses to the UAC.

Each SIP transaction can take a different path through servers in the network. In a typical call, the first request is an `INVITE`, which may traverse many network servers on its way to the callee. The response to the `INVITE` contains a reach address that can be used by the UAC to send subsequent transactions directly to the UAS. Because SIP network servers need not maintain the call state once a transaction is complete, a SIP server has no recollection of the caller or callee. This characteristic facilitates the scalability and reliability of a SIP server, because it can crash and recover (or can swap in a backup), without affecting any of the calls initiated through it. The duration of and amount of state maintained at a server are small compared to those in the global switched telephone network (GSTN), where a switch must maintain the call state for the entire duration of a call. However, a server that wishes to maintain the call state may do so. Through SIP's Route and Record-Route header fields, each proxy individually can insist on being on the signaling path for subsequent transactions. Furthermore, a proxy can change its mind and remove itself from the signaling path later on.

Interestingly, a SIP network server is not required to be *stateful*—that is, to maintain its state—even for the duration of a transaction. A proxy or redirect server can be completely stateless. After it receives a request, it either generates a response or proxies the request, and then forgets everything. The messages themselves contain all the information needed for a stateless proxy to correctly process and route them. This behavior aligns nicely with the Internet datagram architecture, whose packets contain enough information to be individually routed. Furthermore, a stateful proxy can decide to become stateless at any time during a transaction, and the system will still operate correctly. The administrator decides, on a call-by-call basis, whether a proxy will be stateless or stateful. This flexibility of state allows large, central SIP servers to be stateless, but also enables smaller, localized servers to be stateful.

Figure 1 depicts a typical SIP deployment of network servers and the message flow among them. Of the three domains shown (A, B, and C), each has a single SIP server acting as an access point into and out of the networks. Joe, a user agent in domain A, makes a call to Bob, another user. The call invitation is forwarded to the access server of domain A (1), which attempts to find the callee in domains B and C by forking the request.

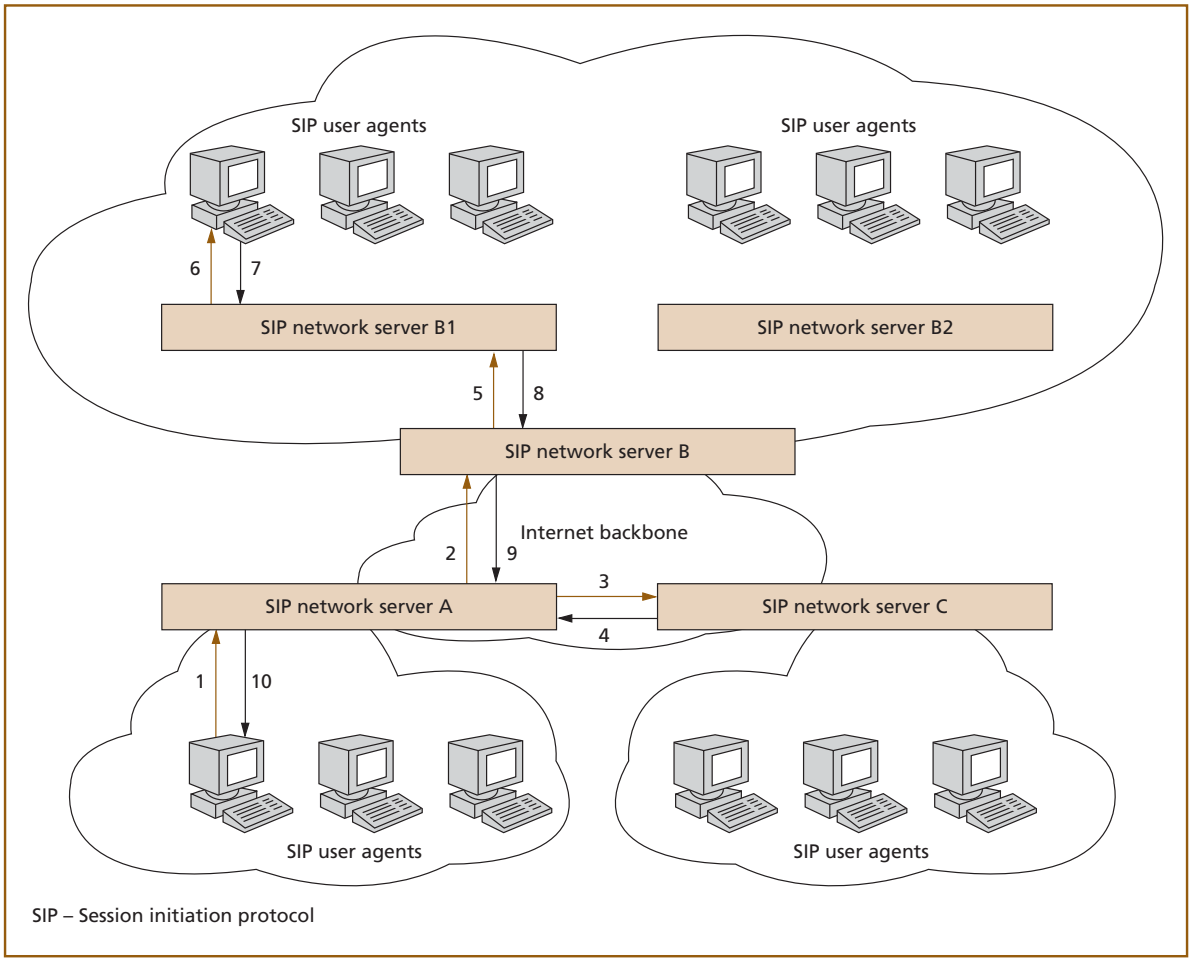


Figure 1.
Typical SIP deployment.

The request arrives at domain C (3), but the user is unknown at this location, so an error response is returned (4). The request at domain B (2), however, is forwarded to a local server internal to domain B (5), where it finally reaches the UAS (6). The response is then returned along the same path to the caller (7,8,9,10).

SIP Messages

A SIP request consists of a request line, header fields, and a message body. The various header fields contain information on call services, addresses, and protocol features. The body, opaque to SIP, can contain anything.

SIP defines several methods, including `INVITE`, `BYE`, `OPTIONS`, `ACK`, `REGISTER`, and `CANCEL`. `INVITE` is used to invite a user to a call. The header fields of an `INVITE` request contain the addresses of the caller and callee, subject of the call, call priority, call routing requests, caller preferences for user location, and desired features of the response, among others. The body of the request contains an opaque description of the media content of the session. Usually, this body is an object described by the session description protocol (SDP),[22] a textual syntax for describing unicast and multicast multimedia sessions. It contains information on codecs, ports, and protocols to be used for sending media to the caller, such as parameters for RTP.[2]

In the case of a broadcast-style multicast-based conference, the body of the request also contains information that describes the content, start and stop times, and administrator of the multicast session. A caller can use this information to invite a callee to participate in an existing multicast backbone (Mbone)[23] session, for example. The amount of information found in SDP is sufficient to allow a caller to begin sending and receiving audio immediately. The response to the `INVITE` contains the media information for the callee. SDP enables a user to indicate the ability to send and receive with multiple audio and video codecs, and SDP can also rank those codecs in preference of usage. Based on experience, this information provides a basic, but sufficient, means for exchanging media capabilities.

Because the body is opaque to SIP, but described using multipurpose Internet mail extension (MIME)[24] syntax and semantics, SIP can use other media description formats besides SDP. These include H.245[25] capability descriptors, Synchronized Multimedia Integration Language (SMIL)[26] presentation descriptions, and Extensible Markup Language (XML)[27] formatted descriptions of a new video codec. As these and other descriptions appear, they are incorporated into SIP as easily as a new image type is incorporated into HTTP.

`REGISTER`, which conveys location information to a SIP server, allows a user to tell a SIP server how to map an incoming address into an outgoing address that will reach that user (or another proxy that knows how to reach that user). The body of a `REGISTER` message can be anything. Currently, researchers are investigating the use of simple scripts to describe the more complex programmatic name translations. This feature, currently planned for standardization within the IETF Internet Protocol Telephony (IPtel) working group, is similar to using service logic for switches. Furthermore, the body of a `REGISTER` response can contain configuration information useful to the user agent. Such information may include speed dial button configurations; additional addresses that allow for private branch exchange (PBX)-like functionality, whereby the server chooses the addresses used by each client; or a call log. Using the multipart MIME formatting rules and the capability negotiation features of HTTP, new types of information can be added as time passes.

Beyond `INVITE` and `REGISTER`, described in the section above, the SIP methods also include:

- `BYE`, which terminates a connection between two users in a conference;
- `OPTIONS`, which solicits information about the capabilities of the callee, but does not set up a call;
- `ACK`, which confirms reliable message exchanges; and
- `CANCEL`, which terminates a pending request, but does not undo a completed call. When received at a UAS, `CANCEL` has no effect if the UAS has already answered the call. If the UAS has not answered, `CANCEL` indicates that it should not bother responding because the call has effectively been canceled. This does not, however, prevent a UAS from answering the call request. It is simply an optimization.

Because SIP is a textual protocol, generation and parsing of its messages are done trivially, particularly with powerful text processing languages such as the Practical Extraction Report Language (Perl). Furthermore, its compliance to RFC 822[28] formatting rules means existing HTTP or SMTP parsers can be used directly for lexicographic analysis of messages. It also vastly simplifies debugging, development cycles, and extensions. **Figure 2** shows a typical SIP `INVITE`.

Addressing and Naming

To be invited and identified, the called party has to be named. SIP chose an e-mail-like identifier of the form user@domain, user@host, user@IP_address, or phone-number@gateway because it is the most common form of user addressing in the Internet. The domain name can be either the name of the host that a user is logged onto at the time, an e-mail address, or the name of a domain-specific name translation service. Addresses of the form phone-number@gateway designate PSTN phone numbers reachable via the named gateway.

Using an e-mail address as a SIP address provides a scalable means by which a UAC can deliver a request

```
INVITE sip:ann@lucent.com SIP/2.0
Via: SIP/2.0/UDP 131.215.131.13;maddr=239.112.3.4;ttl=16
Via: SIP/2.0/TCP 10.0.1.1;received=128.13.44.52
From: John Smith <sip:jsmith@lucent.com>
To: Arun Netravali <sip:ann@lucent.com>
Subject: Raise
Call-ID: 132059753@mypc.domain.lucent.com
Content-Type: application/sdp
CSeq: 4711 INVITE
Content-Length: 187

v=0
o=user1 51633745 1348648134 IN IP4 128.3.4.5
s=Interactive Conference
c=IN IP4 224.2.4.4/127
t=0 0
m=audio 3456 RTP/AVP 0 22
a=rtpmap:22 application/g723.1
```

Figure 2.
Typical SIP `INVITE` ***message.***

to a SIP server, which likely knows how to forward the request to the final callee—the DNS. By performing a series of DNS lookups—such as searching for service (SRV), mail exchange (MX), and address (A) records—the caller can determine the address of a server that has naming authority for all users within the domain.

The use of an e-mail-like identifier also allows SIP addresses to be easily transformed into uniform resource identifiers (URIs),[29] such as `sip:j.doe@example.com`. As such, they can be embedded in Web pages, so that clicking on the link initiates a call to that address, similar to a `mailto`[30] uniform resource location (URL) today.

Integration with Existing Protocols

One of SIP's strengths is its remarkable ability to integrate with existing protocols used on the Internet. In particular, SIP integrates well with the two dominant applications: Web and e-mail.

SIP integrates with the Web on a number of levels. First, SIP carries around MIME content, as does HTTP. This characteristic enables SIP to return Web content as a result of a call invitation. For example, a redirect response to a SIP `INVITE` can contain a HyperText Markup Language (HTML) document or a text document. This document could relate detailed information on alternate places a user might be located (including pictures and sound), or it could contain a form for submitting credit card authorization for the call. It could even return a Java* applet, which accepts input from the caller to determine where a user might be reached. As a result, SIP would integrate extremely well with Web browsers, uniting the Web and telephony to produce new, powerful services.

SIP identifies a user by means of a URL, which can be embedded either in Web pages or in e-mail, as can any other type of URL. Clicking on a URL can initiate calls, just as clicking on a Web link can access a new Web page.

One of the richest features of the Web is its programmability. Web servers can use the common gateway interface (CGI),[31] for example, to create dynamic content, customized for each user. This feature has resulted in many popular Web services, such as access to stock information and movie listings. Because SIP looks like HTTP, we have begun to explore whether CGI can be applied to Internet telephony as well. Our investigations have indicated that CGI can be applied to SIP servers, and by using backwards compatible extensions, can provide a means for rich telephony programming.[32] We have found that telephony services such as call forwarding (and all its variants), mobility, and virtual private network (VPN)—to name a few—are easily implemented with CGI. Furthermore, the wide array of software tools that exist to simplify CGI prototyping can be used for Internet telephony as well.

SIP also integrates well with e-mail and SMTP. A SIP `INVITE` message can be sent by e-mail when all else fails, because a SIP address is identical to an e-mail address. A SIP proxy server can easily reformat a SIP message into an SMTP message, whose formatting is nearly identical, and forward it to an SMTP MTA. This feature integrates voice mail and e-mail to enable call invitations to be delivered via e-mail when a user is not available. In addition, SIP headers can contain mailto URLs, which redirect callers to the user's e-mail, for example.

The real-time streaming protocol (RTSP)[33] allows a client to have VCR-like controls over a media server by instructing the server to play, record, fast-forward, and rewind, among other functions. The client can also specify details about the types of format in which the server should record or play. SIP also integrates well with RTSP.

Like SIP, RTSP is a textual protocol, similar in format to HTTP. RTSP has several uses in IP telephony. First, voice mail is generally considered an important part of telephony. In essence, a voice mail server is nothing more than a media server. RTSP fits naturally as a means for controlling an IP telephony voice mail server. Second, it is often useful to record a call or to play back some prerecorded content into a conference call. RTSP could be used to control the recording and playback in these cases. In fact, RTSP has been engineered to allow a client not only to invite a media server to a session using SIP, but also to control the session using RTSP.

Network Services

Because its strength is in flexible user location and name mapping features, SIP provides a good framework for services such as personal mobility; 700, 800, and 900 services; call screening; forward and transfer; and multiparty calls. (Note that billing in general is outside the scope of SIP, including 700, 800, and 900 services.) Each of these is essentially based on programmed call routing and thus fits well within SIP. Also introduced in SIP are some new services not available on the phone network, such as caller selection, described later in this paper. The sections that follow present just a few of the services possible with SIP.

Personal Mobility

SIP supports advanced personal mobility services, an example of which is shown in **Figure 3**. A user of the system, Bob, maintains an office at a Lucent Technologies location. In addition, Bob is an assistant professor at Columbia University, where he has another lab and office. Bob publishes a single IP telephony phone address for himself: `bob@lucent.com`. When Bob is at Columbia, he sends a `REGISTER` message to the Lucent SIP server (1), listing his Columbia address—`bob@columbia.edu`—as a forwarding address. Once at Columbia, Bob registers both his lab machine (2)—`bob@lab.columbia.edu`—and his office—`bob@office.columbia.edu`—with the Columbia registration server (3). Last time Bob was at Columbia, he set up his lab's computer to automatically forward calls to his Lucent address. Forgetting about this, Bob restarts his user agent in the lab with the same configuration.

Later in the day, `jack@att.com` places a call to `bob@lucent.com`. Using DNS, the caller resolves `lucent.com` to the address of the Lucent SIP server, which receives the call request (4). The server checks its registration and policy databases (5), and decides to forward the request to `bob@columbia.edu`. To do so, it looks up `columbia.edu` in DNS and obtains the address of the main Columbia SIP server. It then forwards the request there (6). As soon as the request arrives, the Columbia server looks up `Bob@columbia.edu` in the registration and policy database (7) and determines that he has two potential means of contact. The server then forks and sends a call request to both the lab and office machines simultaneously (8, 9), causing the office phone to ring. The lab phone receives the request, and according to its outdated configuration, forwards it to Lucent (10). Using the loop detection capabilities in SIP, the Lucent server determines that an error has occurred and returns an error response to the lab machine (11). It, in turn, returns an error code to the Columbia server (12).

In the meantime, Bob answers the phone in his office, sending an acceptance response back to the Columbia server (13). Having now received both responses, the Columbia server forwards the call acceptance back to the Lucent server (14), which forwards

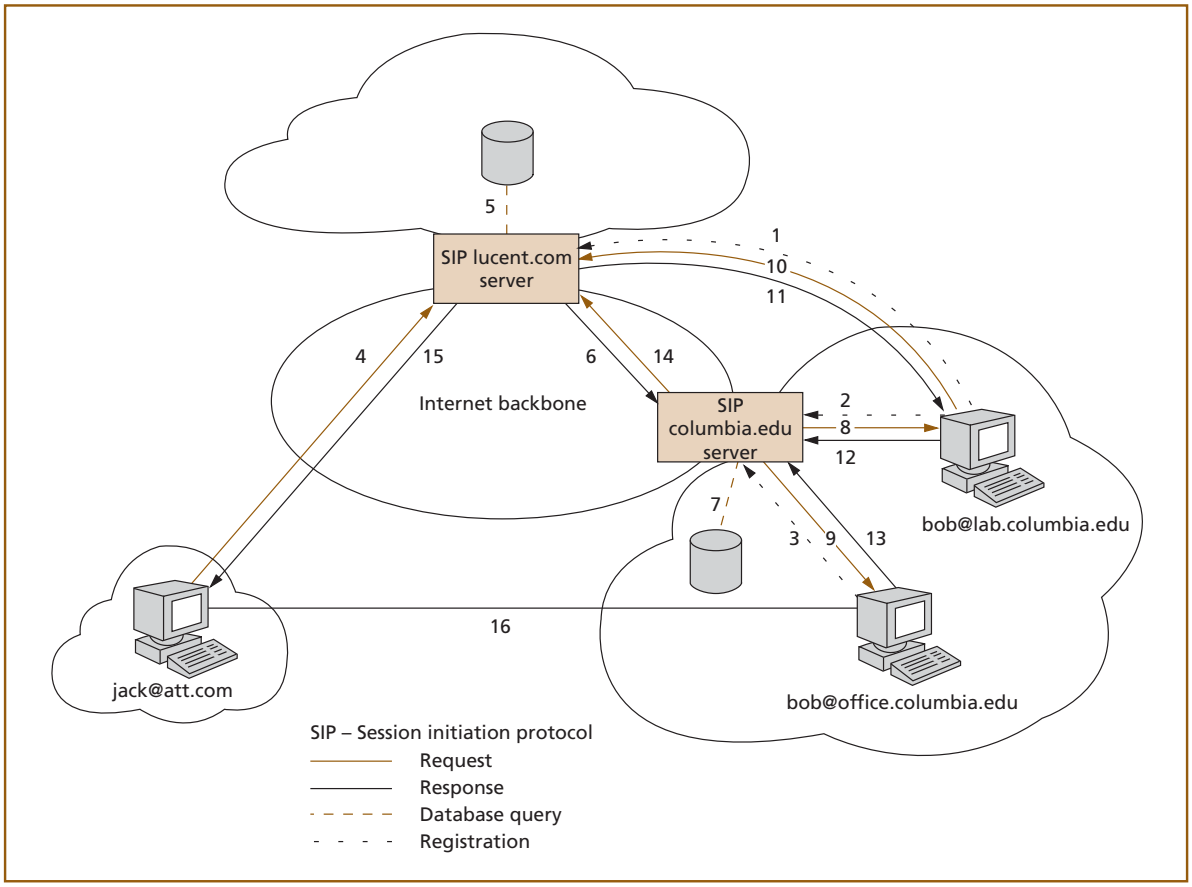


Figure 3.
An example of a SIP mobility service.

the request back to the original caller (15). At this point, the Lucent and Columbia servers can destroy all call states if they so choose. Future call transactions may proceed directly between the caller and Bob without passing through the intermediate servers (16).

The example in Figure 3 illustrates a number of salient features of SIP. First, it shows how a call request can trigger a hunt for a user, hopping among multiple servers until the final target of the call is found. Second, it demonstrates the loop detection features of SIP and, third, how a server can fork requests to speed up the process of contacting the desired user. Finally, it shows that a SIP server is normally used only for the initial call setup (although a server may elect to maintain a call state and remain on the signaling path by using SIP's source routing features). Even in this more limited role, the SIP plays an important part in the delivery of rich mobility services.

Home Phone

One of the more interesting technical challenges for IP telephony is to mimic standard residential phone service. In particular, IP telephony requires the following features:

- When someone calls a particular number, all phones in the home must ring.
- When one of the lines is picked up, all other lines must stop ringing.
- A user can pick up from any other telephone in the home and join an existing call.
- A home can have multiple lines, enabling a user on another handset to initiate a new call while one or more are in progress.
- All users involved in a single call are essentially

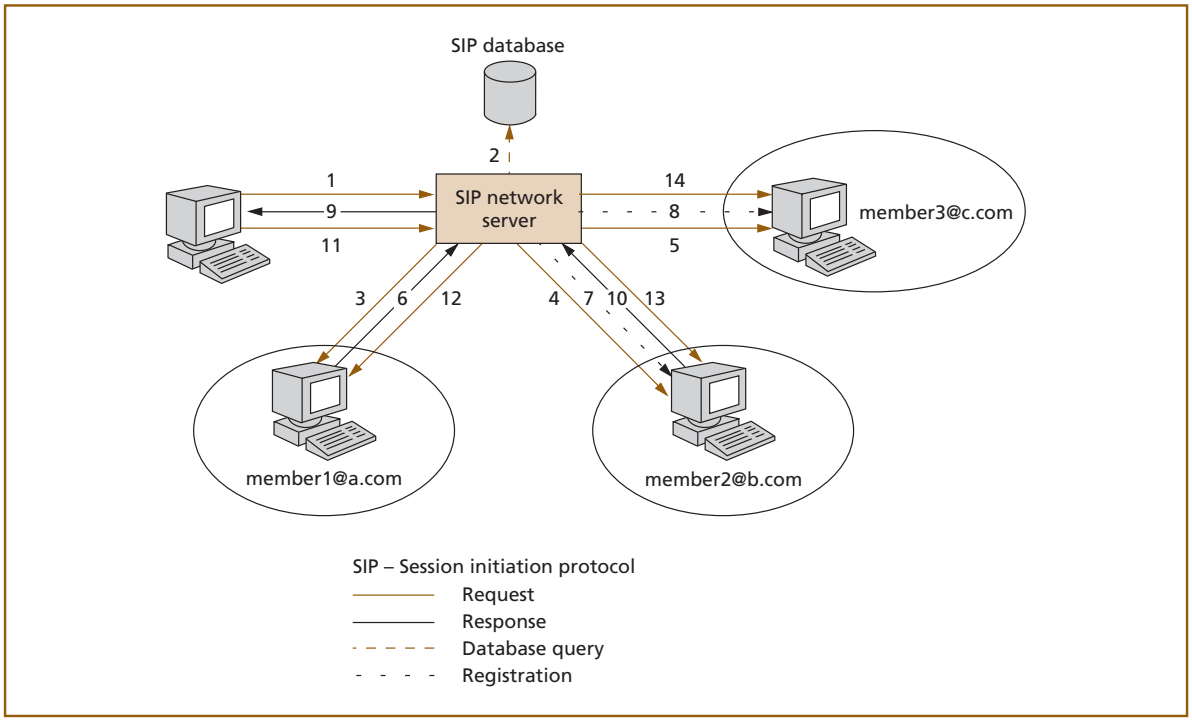


Figure 4.
SIP emulating residential service.

involved in a multiparty conference call, and are thus able to hear each other.

SIP easily emulates this basic service, as shown in **Figure 4**.

Here, the caller sends a SIP `INVITE` to `smith_family@isp.com`. The `INVITE` is sent to the main server for ISP, the Internet service provider (1). The ISP consults its policy database (2), which indicates that the called address is a residential line. The database also includes a list of contact addresses, each of which constitutes a single extension on that line. The proxy server thus forks and sends out three `INVITE` requests, (3, 4, 5), one to each address. Furthermore, the server modifies the SDP in the message, indicating that a multicast address is being used for media exchange.

The lines ring at each address. A user picks up on the first line, which causes a call acceptance to be sent back to the server (6). According to the SIP specifications, when a forking proxy server receives a call acceptance on any one of its branches, it should send a `CANCEL` request on all unanswered branches. The proxy then sends out a `CANCEL` request on the other two branches (7, 8). Because neither address has yet answered, it cancels the call request, effectively terminating the call and keeping the phones from ringing.

The server forwards the call acceptance back to the caller (9). It also acts as a multicast to unicast bridge, forwarding media received on the multicast group back to the caller, and vice versa. The server also lists itself in the call acceptance as the main point of contact for continued signaling messages.

Later, one of the other lines is picked up, forwarding another call acceptance back to the server (10). This acceptance is acknowledged by the server. Using the multicast group, the new participant is now part of the call. As soon as the caller hangs up, a `BYE` message is sent to the server (11), which in turn sends the message to all three branches (12, 13, 14). As a result, the two currently active lines terminate and the unanswered line destroys the state associated with the call.

This example further illustrates the power of

forked call requests, combined with the utility of the CANCEL request. The behavior described for the SIP clients in the example described above is standard procedure, not specific for residential service. Clients of either residential or traditional business service can use the same piece of software. Only the server in the above example required special programming to direct it towards residential behavior (although this behavior was still compliant with the SIP specification). The same mechanisms can be used equally well for ACD services.

Outsourced Call Screening

The use of SIP proxy servers enables various call logic services to be outsourced to third-party vendors. Consider the case of outgoing call screening services. A parent wishes to prevent her teenage son from making calls to various adult telephone numbers. Instead of trying to keep track of the numbers of such services, and programming filters into the software in the home, the parent subscribes to a filtering service, which keeps track of such numbers automatically.

Figure 5 depicts how SIP may be used for this kind of service. The user's home software is configured to automatically forward all call requests to service831@we_do_filters.com. Whenever a call is made from the home, the call is forwarded to the server of the filtering provider (1). Once there, the service checks the URI of the incoming requests and notices that it is service 831, adult line filtering. The server performs a database lookup (2) and finds that the number called is in the database. The server then returns a call rejection to the caller (3), including a phrase that indicates this is a forbidden number. Had the called number not been in the database, the server would have proxied the call to the listed number, allowing it to complete normally.

Caller Selection

SIP also allows for multicast signaling and a new feature, *caller selection*, which enables the call initiator to choose whom to talk to when multiple parties answer a call. **Figure 6** demonstrates an example that uses caller selection and multicast.

In this scenario, there are no network servers. The caller, an employee of a small company, needs computer assistance, so he sends a SIP INVITE message to sysadmins@company.com. The caller's machine has

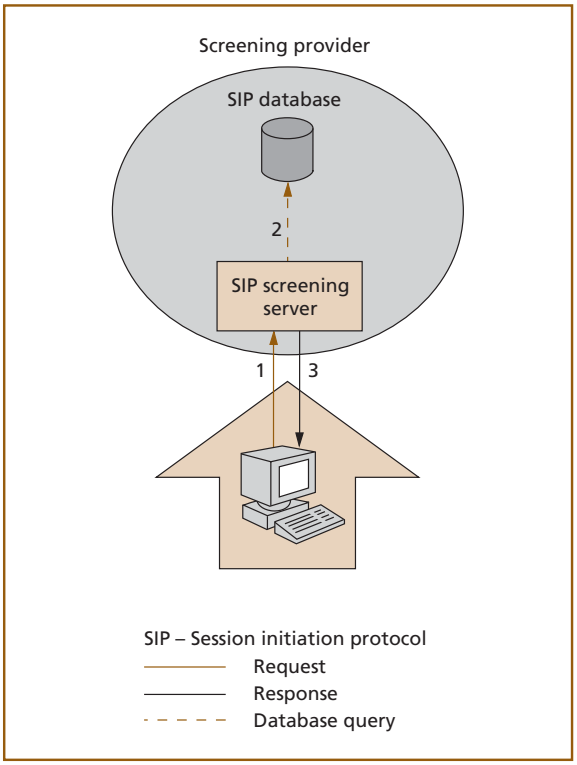


Figure 5.
SIP screening service.

been preconfigured to send messages destined for this address to a particular multicast address. All of the system administrators in the company have configured their software to listen for call requests on this address. All three system administrators receive the INVITE request (1). One of them answers immediately, and the response is also multicast (2). The software of the other system administrators reacts as if a CANCEL had been sent. Their lines stop ringing, and the call between the caller and the first administrator progresses. Later on, a second administrator answers, also sending a call acceptance on the multicast group (3). Because SIP allows multiple call acceptances to respond to a single request, the caller decides to accept the second response as well. Using SIP's third-party call control mechanisms, the caller sets up a multicast media conference among all three.

When the second call acceptance arrived, the caller could have taken any one of a number of actions. These include accepting the call, but not set-

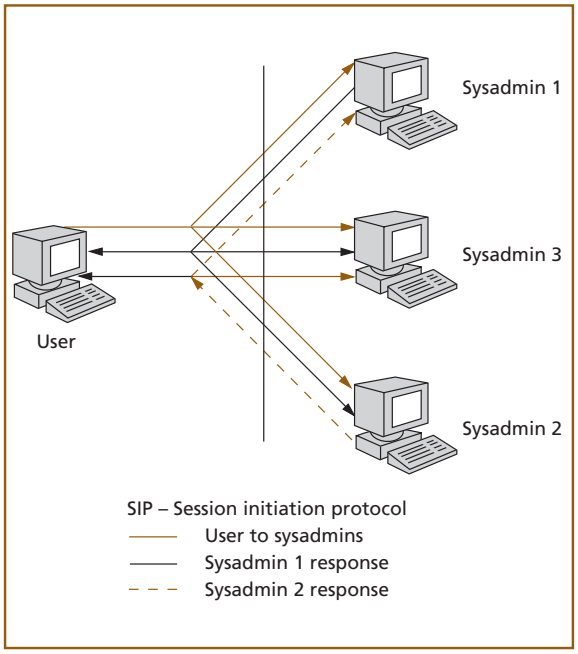


Figure 6.
An example of caller selection and multicasting.

ting up a conference (in which case the caller is effectively connected to two calls), hanging up on either call or both, or redirecting the second acceptor to voice mail. Many other possibilities exist as well. This example demonstrates a simple ACD type of service with fully distributed mechanisms. No call distribution server was required.

Client Services

As the earlier examples have illustrated, SIP network servers are generally used in services related to address resolution, naming, user location, forwarding, and policy. Telephony services encompass more than just these, of course. They include features such as multiparty conferencing, transfer, hold, and mute. An Internet environment can provide these services directly in user agent software, without the need for network server support. However, since a proxy is just a protocol server and a protocol client back to back, a network server can also provide these services using the same end-to-end mechanisms. Because the network server is a unified mechanism, it can easily deploy services at any point in the network. This type of flexibility allows SIP to be used in environments ranging from the rightly controlled and managed PBX to fully distributed, serverless networks.

SIP supports multiparty and transfer services largely through two end-to-end headers, `Also` and `Replaces`. When included in either a request or a response, these headers instruct the recipient to either place a call to the parties listed in the header or terminate it, in that order. This simple mechanism provides a framework for basic third-party call control. In turn, it may be easily used to construct a variety of transfer and multiparty services.[34]

As another example of SIP's ability to support new services, the PSTN and IP Internetworking (PINT) working group of the IETF is in the process of standardizing a protocol for click-to-dial and click-to-fax services. Click-to-dial allows a user to click on an icon while Web browsing, and receive a phone call from a service representative on the GSTN. The protocol that provides this service allows a Web server to pass third-party call control commands to a PSTN-enabled server, such as a PBX or a service control point (SCP), as shown in **Figure 7**. The requirements a protocol must fulfill to accomplish this include flexible addressing and naming, authentication, encryption, extensibility, and third-party call control primitives. Because SIP is ideal for this scenario, the working group is standardizing some basic SIP extensions to provide the service.

Key Benefits

SIP offers a number of key benefits, which we discuss here in more detail. These include simplicity, extensibility, modularity, scalability, integration, and enhanced services.

Simplicity

SIP is a very simple protocol; its specification is just 99 pages long. SIP has only 42 headers, each with a small number of values and parameters. A basic, but interoperable SIP Internet telephony implementation can get by with four headers (`To`, `From`, `Call-ID`, and `CSeq`) and three request methods (`INVITE`, `ACK`, and `BYE`), and it is small enough to be assigned as a homework programming problem in the Advanced Internet Services class taught at Columbia University. A fully functional SIP client agent with a graphical

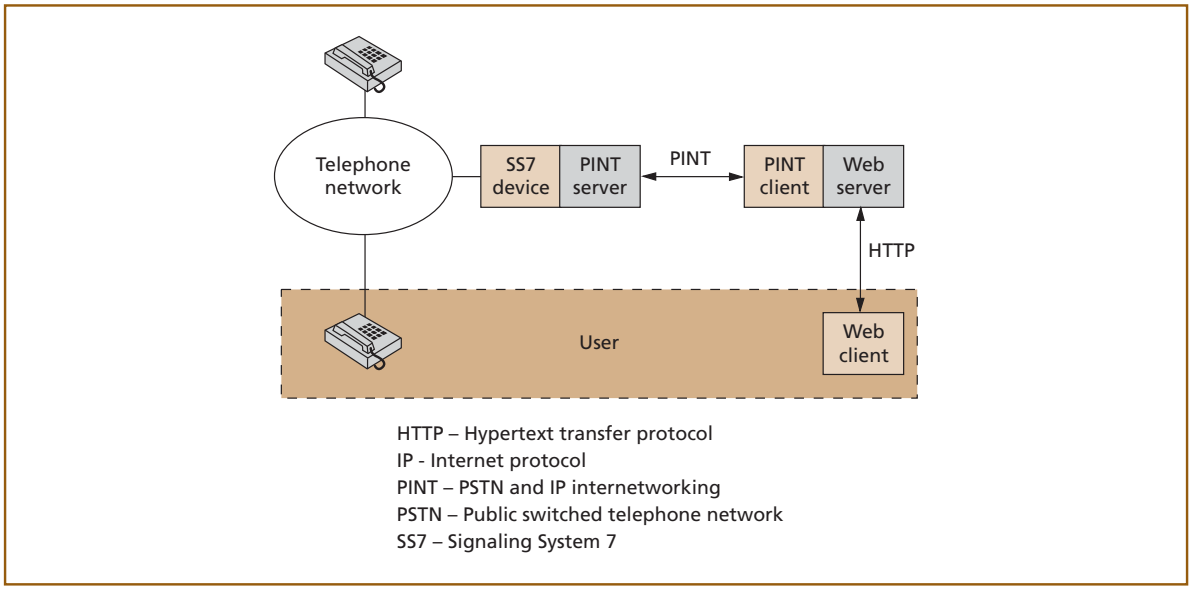


Figure 7.
Click to dial service and PINT.

user interface has already been implemented in just two months.

Because SIP encodes its messages as text, parsing and generation are simple. This is especially true when done with powerful text processing languages such as Perl. The textual encoding also simplifies debugging, allowing manual entry and perusing of messages. Its similarity to HTTP also allows for code reuse; existing HTTP parsers can be quickly modified for SIP usage.

Extensibility

Extensibility is a key metric for measuring an IP telephony signaling protocol. Currently, telephony is one of the largest, economically important, widespread, critical services, but over the long run Internet telephony is likely to supplant the existing circuit-switched infrastructure developed to support it. As with any heavily used service, the features provided evolve over time as new applications are developed, introducing problems in compatibility among versions. The Internet is an open, distributed, and evolving entity where we can expect extensions to IP telephony protocols to be widespread and uncoordinated, making it critical to build in powerful extension mechanisms from the outset.

SIP has learned the lessons of HTTP and SMTP (both of which are widely used protocols that have evolved over time), and has built in a rich set of extensibility and compatibility functions. By default, unknown headers and values are ignored. Using the `Require` header, clients can indicate named feature sets that the server must understand. When a request arrives at a server, it checks the list of named features in the `Require` header. If any of them are not supported, the server returns an error code and lists the set of features it does understand. The client can then determine the problematic feature and fall back to a simpler operation. The feature names are based on a hierarchical name space, and new feature names can be registered with the Internet Assigned Numbers Authority (IANA). SIP allows any developer to create new features within SIP and then simply register a name for them. Compatibility is still maintained across different versions.

To further enhance extensibility, numerical error codes are hierarchically organized, as in HTTP. Each of six basic classes of error codes is identified by the hundreds digit in the response code. Basic protocol operation is dictated solely by the class, and terminals need only understand the class of the response. The other

digits provide usually useful, but not critical, additional information. This hierarchical model allows developers to add features by defining semantics for the error codes in a class, while maintaining compatibility.

Using textual encoding—such as `To`, `From`, and `Subject`—to describe the header fields keeps their meaning self-evident. As new header fields are added in various different implementations, developers in other corporations can determine their usage just from the name, and add support for the field. This kind of distributed, documentation-less standardization is common in SMTP, which has evolved over the years. In addition, textual encoding allows developers to add new header fields, parameters, or attributes anywhere in the message without breaking existing parsers.

Because SIP is similar to HTTP, mechanisms being developed for HTTP extensibility can also be used in SIP. Among these are the protocol extensions protocol (PEP),[35] which contains pointers to the documentation for various features within the HTTP messages themselves.

Modularity

Another aspect of extensibility is modularity. Internet telephony requires many different functions, some of which have been mentioned previously. One can be certain that mechanisms for accomplishing these functions will evolve over time (especially those that relate to QoS). It is critical to apportion these functions to separate, modular, orthogonal components, which over time can be swapped in and out of software and systems used for providing IP telephony. It is also critical to use separate, general protocols for each function, to allow it to be duplicated in other applications with ease. For example, it is more efficient to have a single application-independent QoS mechanism than it is to invent a new QoS protocol or mechanism for each application.

SIP is very modular. It encompasses basic call signaling (initiation, termination, and change), user location, and basic registration (essential for user location). Third-party call control, used for transfer and multiparty services, is in a single SIP extension. Quality of service, directory accesses, service discovery, session content description, and conference control are all orthogonal and reside in separate protocols.

A key feature of SIP is its ability to separate the notion of a session from the protocol used to invite a user to a session. SIP just issues invitations; it does not know anything about the session itself. Protocols such as SDP (which describes media sessions) are used for this purpose. SIP can also be used to invite users to control sessions, broadcast television sessions, and document editing sessions—even virtual reality sessions—once description formats for these sessions have been developed. Proxy servers do not need to know or understand anything about these sessions. For this very reason, a proxy server deployed today will still work with future applications. This kind of modularity has allowed the Web to flourish. For example, although HTTP was initially used only to carry HTML, it now carries a wide variety of content types. Had HTTP only carried HTML, there would surely be no Web.

Interestingly, the payload carried by SIP can be a program, such as a Java applet or a Tcl code fragment. These programs can dynamically define what the session is, depending on their output, making SIP a powerful tool for future intelligent network (IN) applications, where programmability is key.

Scalability

We can observe scalability on a number of different levels, including:

- *Domains*. To provide a wide area of operation, SIP leverages off DNS, as well as off powerful routing protocols such as BGP. End systems can be located anywhere on the Internet, without requiring the use of additional name resolution services.
- *Server processing*. In SIP, a transaction through several servers and gateways can be either stateful or stateless. In the case of UDP, no connection state is even required, meaning that large backbone servers can be based on UDP and operate in a stateless fashion, reducing memory requirements and improving scalability. SIP servers at network edges, or within the enterprise, can be stateful, allowing them to offer more complex services. This model of simplicity within the core and complexity at the periphery has been the cornerstone of Internet scalability for some time.

- *Conference sizes*. SIP scales to all different conference sizes. Because there is no requirement for a central multipoint controller, conference coordination can be fully distributed or centralized, at the discretion of the implementer. SIP also works for large, broadcast-style conferences. In fact, originally SIP was used to invite users to participate in Mbone conferences. For this reason, SIP is implemented in the popular mbone session directory tool, sdr, used to determine the sessions currently being broadcast on the Mbone.[36]

Integration

Another important feature of SIP is its ability to integrate well with the Web, e-mail, streaming media applications and protocols, and other networking services, to name a few. The power of the Internet is in its support for a vast array of applications. An Internet telephony protocol should therefore strive to integrate well with other Internet applications, particularly the Web.

Conclusion

In this paper, we have presented the session initiation protocol (SIP), used for the initiation, control, and termination of multimedia conferences. It builds on the strengths of HTTP and SMTP, mapping telephony services into simple request-response transactions. The strength of SIP lies in its ability to support rich mobility, forwarding, address resolution, and naming services. We have shown how SIP can be used for such things as personal mobility, call screening, residential line services, and ACD. We have briefly mentioned the third-party call control services available in SIP.

A SIP server, user agent, and gateway are all currently under development within Bell Labs. Prototypes of all three were demonstrated in June 1998 at the SUPERCOMM trade show. The SIP client is a Java-based application that uses the Java telephony application programming interface (JTAPI)[37] to rapidly develop other SIP applications. The server, a multiplatform software toolkit, incorporates support for hierarchical policy databases to guide routing decisions, which can be used to construct many of the services described in this paper. The SIP gateway interfaces to the Lucent PacketStar™ IP Services Platform, available as a new product from Lucent. The platform provides a common framework on which call services for SIP, H.323, integrated services digital network (ISDN), and "plain old telephone service" (POTS) can be constructed.

References

1. H. Schulzrinne, "Re-engineering the telephone system," *Proc. of IEEE Singapore Intl. Conf. on Networks (SICON) 1997: The Next Millennium*, Singapore, Apr. 14–17, 1997, pp. 261–275.
2. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications, Request for Comments (Proposed Standard) 1889," IETF, Jan. 1996.
3. B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP)—version 1 functional specification, Request for Comments (Proposed Standard) 2205," IETF, Oct. 1997.
4. B. Braden and L. Zhang, "Resource ReSerVation protocol (RSVP) version 1 message processing rules, Request for Comments (Proposed Standard) 2209," IETF, Oct. 1997.
5. P. P. Pan and H. Schulzrinne, "Yessir: A simple reservation mechanism for the Internet," *Proc. Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, U.K., July 1998.
6. S. Blake, Y. Bernet, J. Binder, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss, "A Framework for Differentiated Services," Internet Draft, IETF, Oct. 1998. http://www.ietf.org/internet-drafts/draft-ietf-diffserv-framework-01.txt
7. C. Rigney, A. Rubens, W. Simpson, and S. Willens, "Remote authentication dial in user service (RADIUS), Request for Comments (Proposed Standard) 2138," IETF, Apr. 1997.
8. C. Rigney, "RADIUS accounting, Request for Comments (Informational) 2139," IETF, Apr. 1997.
9. A. Rubens and P. Calhoun, "DIAMETER base protocol," Internet Draft, IETF, May 1998. http://www.ietf.org/internet-drafts/draft- calhoun-diameter-07.txt
10. J. Rosenberg and H. Schulzrinne, "Internet telephony gateway location," *Proc. of the Conf. on Comp. Commun. (IEEE INFOCOM '98)*, San Francisco, Mar. 29–Apr. 2, 1998.
11. J. Rosenberg and H. Schulzrinne, "A Framework for a Gateway Location Protocol,"

Internet Draft, IETF, Oct. 1998. http://www.ietf.org/internet-drafts/draft-ietf-iptel-gwloc-framework-01.txt
12. T. Howes, S. Kille, and M. Wahl, "Lightweight directory access protocol (v3), Request for Comments (Proposed Standard) 2251," IETF, Dec. 1997.
13. "Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service," ITU-T Rec. H.323, Geneva, May 1996.
14. M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Internet Draft, IETF, Sept. 1998. http://www.ietf.org/internet-drafts/draft-ietf-mmusic-sip-11.txt
15. H. Schulzrinne and J. Rosenberg, "A comparison of SIP and H.323 for Internet telephony," *Proc. Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, U.K., July 1998.
16. J. Postel, "Simple mail transfer protocol, Request for Comments (Standard) STD 10, 821," IETF, Aug. 1982.
17. J. Klensin and D. Mann, "Simple mail transfer protocol," Internet Draft, IETF, May 1998. http://www.ietf.org/internet-drafts/draft-ietf-drums-smtpupd-08.txt
18. R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee, "Hypertext transfer protocol HTTP/1.1, Request for Comments (Proposed Standard) 2068," IETF, Jan. 1997.
19. P. Mockapetris, "Domain names—concepts and facilities, Request for Comments (Standard) STD 13, 1034," IETF, Nov. 1987.
20. P. Mockapetris, "Domain names—implementation and specification, Request for Comments (Standard) STD 13," 1035, IETF, Nov. 1987.
21. Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4), Request for Comments (Draft Standard) 1771," IETF, Mar. 1995.
22. M. Handley and V. Jacobson, "SDP: session description protocol, Request for Comments (Proposed Standard) 2327," IETF, Apr. 1998.
23. H. Eriksson, "MBone: the multicast backbone," *Proc. of the Intl. Networking Conf. (INET)*, San Francisco, Internet Society, Aug. 1993, pp. CCC1–CCC5.
24. N. Borenstein and N. Freed, "MIME (multipurpose Internet mail extensions): Mechanisms for specifying and describing the format of Internet message bodies, Request for Comments (Proposed Standard) 1341," IETF, June 1992.
25. "Control protocol for multimedia communication," ITU-T Rec. H.245, Aug. 1997.
26. World Wide Web Consortium, "Synchronized multimedia integration language (smil) 1.0 specification, Rec. PR-smil-19980409," Apr. 1998.
27. World Wide Web Consortium, "Extensible markup language (xml) 1.0, Rec. REC-xml-19980210," Feb. 1998.
28. D. Crocker, "Standard for the format of ARPA internet text messages, Request for Comments (Standard) STD 11, 822," IETF, Aug. 1982.
29. T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax, Request for Comments 2396," IETF, Aug. 1998.
30. P. Hoffman, L. Masinter, and J. Zawinski, "The mailto URL scheme, Request for Comments (Proposed Standard) 2368," IETF, July 1998.
31. D. Robinson and K. Coar, "The WWW common gateway interface version 1.1," Internet Draft, IETF, May 1998. http://www.ietf.org/internet-drafts/draft-coar-cgi-v11-01.txt
32. J. Lennox, J. Rosenberg, and H. Schulzrinne, "A Common Gateway Interface for SIP," Internet Draft, IETF, Nov. 1998. http://www.ietf.org/internet-drafts/draft-lennox-sip-cgi-00.txt
33. H. Schulzrinne, R. Lanphier, and A. Rao, "Real time streaming protocol (RTSP), Request for Comments (Proposed Standard) 2326," IETF, Apr. 1998.
34. H. Schulzrinne and J. Rosenberg, "Signaling for Internet telephony," *Intl. Conf. on Network Protocols (ICNP)*, Austin, Texas, Oct. 1998.
35. D. Connolly, H. Nielsen, R. Khare, and E. Prud'hommeaux, "PEP—An extension mechanism for HTTP," Internet Draft, IETF, Dec. 1997. http://www.ietf.org/internet-drafts/draft-ietf-http-pep-05.txt
36. "Session Directory," University College, London. http://www-mice.cs.ucl.ac.uk/multimedia/software/sdr/
37. Sun Microsystems, "The java telephony API." http://www.javasoft.com/products/jtapi/

(Manuscript approved December 1998)

***Trademark**
Java is a trademark of Sun Microsystems.

HENNING G. SCHULZRINNE received a B.S. degree from the Darmstadt University of Technology in Germany, an M.S. degree from the University of Cincinnati in Ohio, and a Ph.D. from the University of Massachusetts in Amherst, all in electrical engineering. An associate professor of computer science and electrical engineering at Columbia University in New York City, Dr. Schulzrinne also works as a consultant in the Wireless Networking Department at Bell Labs in Holmdel, New Jersey. His research interests include Internet telephony, Internet multimedia control and transport, and performance evaluation.

JONATHAN D. ROSENBERG, a member of technical staff in the High Speed Networks Research Department at Bell Labs in Holmdel, New Jersey, conducts research on technologies related to multimedia communications on the Internet, including transport and error recovery, signaling, architectures, protocols, and service creation. Mr. Rosenberg received B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology in Cambridge and is continuing his studies in the same field as a Ph.D. candidate at Columbia University in New York City. ◆

Next: SIP Protocol Details **Up:** Session Directories, Advertisement and **Previous:** Session Announcement Protocol (SAP)

Section Initiation Protocol (SIP)

Status:IETF Proposed Standard

In the near future, if you make a telephone call, it's quite likely that the Session Initiation Protocol will be what finds the person you're trying to reach and causes their phone to ring. SIP is all about calling people and services.

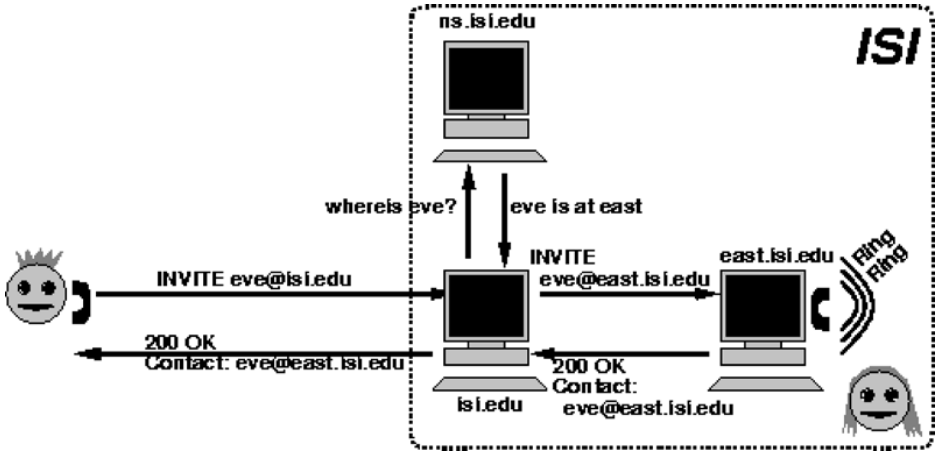
The most important way that SIP differs from making an existing telephone call (apart from that it's an IP-based protocol) is that you may not be dialing a number at all. Although SIP can call traditional telephone numbers, SIP's native concept of an address is a SIP URL, which looks very like an email address.

When you email *joe@gadgets.com*, as a user you have no real idea which computer at *gadgets.com* your email will eventually reach, or indeed if Joe is currently contracting at another company, your message may even be forwarded there. All you wanted to do was to send a message to Joe.

The SIP authors believe that making a telephone or multimedia call should be similar - you want to call Joe, but you don't really care exactly which ``phone'' Joe uses to answer your call. SIP solves this problem by combining user location with the request to set up the call. It might seem that these should be separate, but this is not the case - the routing the call takes may depend on who is trying to make the call. If Joe's sister calls him while he's out at lunch, he might want the call to be routed to his cell-phone, but if his boss makes the same call, the call gets routed to his voice-mail. For privacy reasons, performing user location without actually making the complete call is simply unacceptable to many people.

So how does SIP do this? SIP makes extensive use of proxy servers, each of which looks at the call request, looks up whatever local information it has about the person being called (i.e., the *callee*), performs any security checks it's been asked by the callee or her organisation to make, and then routes the call onward.

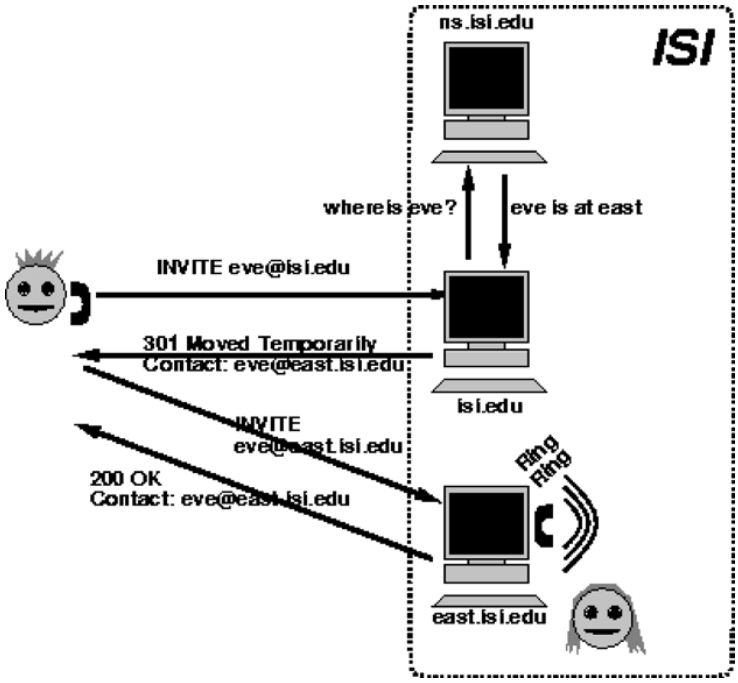
Figure 6.3: SIP Request Being Relayed



When I call *jane@euphoric-state.edu*, my computer (or phone exchange) looks up *euphoric-state.edu* in the Domain Name System (DNS) and looks for a SIP service record giving the address of the SIP server (in this case *sip.euphoric-state.edu*) for Euphoric State University. It then sends the SIP request to that machine. At *sip.euphoric-state.edu*, the server consults a database and discovers that Jane is a staff member in the Computer Science department, and that the SIP server for CS is *sipgw.cs.esu.edu*. It then sends the SIP request on to *sipgw.cs.esu.edu*, which again consults a database. This new database happens to be built dynamically by SIP clients registering when people log on. It turns out that Jane is a professor, and computer science professors never adopt new technology early so her workstation is not capable of multimedia conferencing. Instead *sipgw.cs.esu.edu* routes the call to her regular telephone and it acts as a gateway into the department PBX.

The example above uses proxies that *relay* the SIP call. This is illustrated in figure 6.3. Relaying is often performed when the gateway or firewall to a site wishes to hide any search that goes on internally from the caller's machine.

Figure 6.4: SIP Request Being Redirected

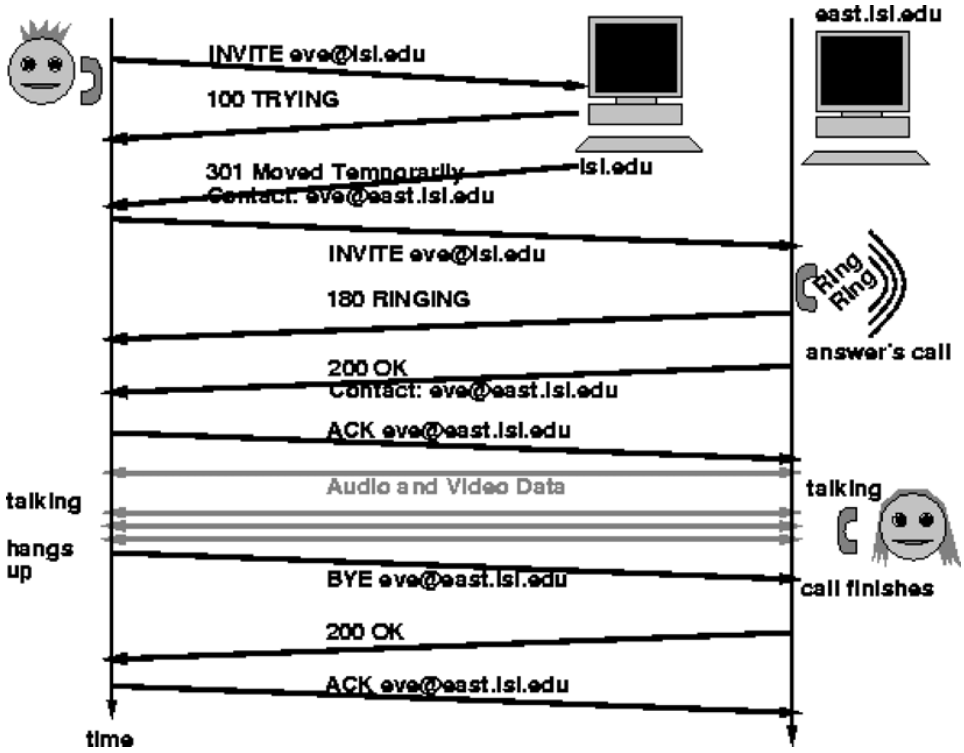


An alternative to relaying is *redirection*, which is illustrated in figure 6.4. Redirection makes the caller's machine do any additional work, so it makes life easier for the proxy server. This might be appropriate when the callee has moved outside of the local network of the proxy.

Of course if you do know the machine the callee is using, you can also direct the request there directly without going through any proxies, but some sites may insert a proxy in the path anyway because the proxy is on their firewall machine.

Figure 6.5 shows a complete SIP call, with the arrows indicating SIP requests and responses passing backward and forward. In this case the SIP *INVITE* request for *eve@isi.edu* is sent to *isi.edu*, but is then redirected to *eve@east.isi.edu*. A new *INVITE* request is then made, which succeeds in finding Eve, and causing her phone to ring, which is reported back to the caller. When Eve picks up the phone, the request is completed with a ``*200 OK*'' response, which is in turn acknowledged, and a voice call is set up. Multimedia information is now exchanged using RTP for as long as they continue to chat. After they've finished their conversation, the caller hangs up, and this causes a SIP *BYE* request to be sent, which proceeds to close down the call.

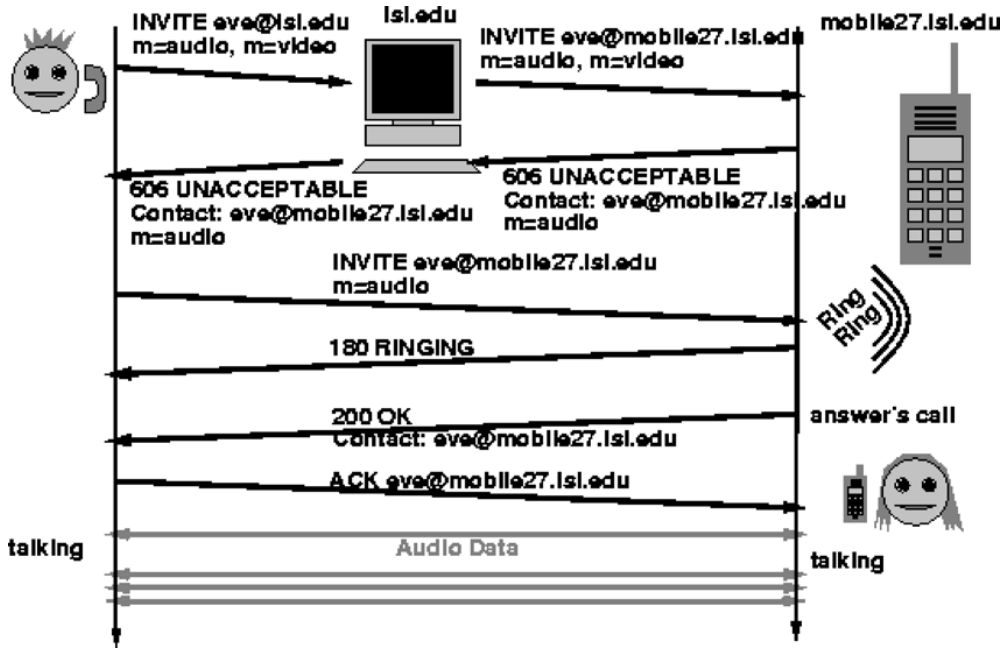
Figure 6.5: Complete SIP Call



SIP can set up many types of call, not just telephone calls, because it was originally designed for setting up multimedia conferences. However, the world can be a very heterogeneous place, and the person making a call may not be able to predict what equipment is available at the other end to receive the call. For example, I may make a video-phone call to you, and although you may have an equivalent video-phone on your desk, it happens that you're out of the office when I call. Instead my call is redirected to your cell-phone, which unfortunately doesn't yet support video-conferencing, but it could perfectly well support an audio call.

This is illustrated in figure 6.6. The original call attempt requests an audio/video call, and is relayed to *mobile27.isi.edu*. *Mobile27* is a cell-phone, and so it rejects the call with a ``*606 Not Acceptable*'' response code indicating that an audio-only call would be possible. A new call attempt is made suggesting an audio call, and then everyone is happy.

Figure 6.6: SIP ``negotiation'' of media parameters



- SIP Protocol Details
- SIP Reliability
- SIP: In Summary

Next: SIP Protocol Details **Up:** Session Directories, Advertisement and **Previous:** Session Announcement Protocol (SAP)

Jon CROWCROFT
1998-12-03

Network Working Group
Request for Comments: 2543
Category: Standards Track

M. Handley
ACIRI
H. Schulzrinne
Columbia U.
E. Schooler
Cal Tech
J. Rosenberg
Bell Labs
March 1999

SIP: Session Initiation Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

The IESG intends to charter, in the near future, one or more working groups to produce standards for "name lookup", where such names would include electronic mail addresses and telephone numbers, and the result of such a lookup would be a list of attributes and characteristics of the user or terminal associated with the name. Groups which are in need of a "name lookup" protocol should follow the development of these new working groups rather than using SIP for this function. In addition it is anticipated that SIP will migrate towards using such protocols, and SIP implementors are advised to monitor these efforts.

Abstract

The Session Initiation Protocol (SIP) is an application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants. These sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution. Members in a session can communicate via multicast or via a mesh of unicast relations, or a combination of these.

SIP invitations used to create sessions carry session descriptions which allow participants to agree on a set of compatible media types. SIP supports user mobility by proxying and redirecting requests to the user's current location. Users can register their current location. SIP is not tied to any particular conference control protocol. SIP is designed to be independent of the lower-layer transport protocol and can be extended with additional capabilities.

Table of Contents

1 Introduction

1.1 Overview of SIP Functionality

The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify and terminate multimedia sessions or calls. These multimedia sessions include multimedia conferences, distance learning, Internet telephony and similar applications. SIP can invite both persons and "robots", such as a media storage service. SIP can invite parties to both unicast and multicast sessions; the initiator does not necessarily have to be a member of the session to which it is inviting. Media and participants can be added to an existing session.

SIP can be used to initiate sessions as well as invite members to sessions that have been advertised and established by other means. Sessions can be advertised using multicast protocols such as SAP, electronic mail, news groups, web pages or directories (LDAP), among others.

SIP transparently supports name mapping and redirection services, allowing the implementation of ISDN and Intelligent Network telephony subscriber services. These facilities also enable personal mobility. In the parlance of telecommunications intelligent network services, this is defined as: "Personal mobility is the ability of end users to originate and receive calls and access subscribed telecommunication services on any terminal in any location, and the ability of the network to identify end users as they move. Personal mobility is based on the use of a unique personal identity (i.e., personal number)." [1]. Personal mobility complements terminal mobility, i.e., the ability to maintain communications when moving a single end system from one subnet to another.

SIP supports five facets of establishing and terminating multimedia communications:

User location: determination of the end system to be used for communication;

User capabilities: determination of the media and media parameters to be used;

User availability: determination of the willingness of the called party to engage in communications;

Call setup: "ringing", establishment of call parameters at both called and calling party;

Call handling: including transfer and termination of calls.

SIP can also initiate multi-party calls using a multipoint control unit (MCU) or fully-meshed interconnection instead of multicast. Internet telephony gateways that connect Public Switched Telephone Network (PSTN) parties can also use SIP to set up calls between them.

SIP is designed as part of the overall IETF multimedia data and control architecture currently incorporating protocols such as RSVP (RFC 2205 [2]) for reserving network resources, the real-time transport protocol (RTP) (RFC 1889 [3]) for transporting real-time data and providing QOS feedback, the real-time streaming protocol (RTSP) (RFC 2326 [4]) for controlling delivery of streaming media, the session announcement protocol (SAP) [5] for advertising multimedia sessions via multicast and the session description protocol (SDP) (RFC 2327 [6]) for describing multimedia sessions. However, the functionality and operation of SIP does not depend on any of these protocols.

SIP can also be used in conjunction with other call setup and signaling protocols. In that mode, an end system uses SIP exchanges to determine the appropriate end system address and protocol from a given address that is protocol-independent. For example, SIP could be used to determine that the party can be reached via H.323 [7], obtain the H.245 [8] gateway and user address and then use H.225.0 [9] to establish the call.

In another example, SIP might be used to determine that the callee is reachable via the PSTN and indicate the phone number to be called, possibly suggesting an Internet-to-PSTN gateway to be used.

SIP does not offer conference control services such as floor control or voting and does not prescribe how a conference is to be managed, but SIP can be used to introduce conference control protocols. SIP does not allocate multicast addresses.

SIP can invite users to sessions with and without resource reservation. SIP does not reserve resources, but can convey to the invited system the information necessary to do this.

1.2 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [10] and indicate requirement levels for compliant SIP implementations.

1.3 Definitions

This specification uses a number of terms to refer to the roles played by participants in SIP communications. The definitions of client, server and proxy are similar to those used by the Hypertext Transport Protocol (HTTP) (RFC 2068 [11]). The terms and generic syntax of URI and URL are defined in RFC 2396 [12]. The following terms have special significance for SIP.

Call: A call consists of all participants in a conference invited by a common source. A SIP call is identified by a globally unique call-id (Section 6.12). Thus, if a user is, for example, invited to the same multicast session by several people, each of these invitations will be a unique call. A point-to-point Internet telephony conversation maps into a single SIP call. In a multiparty conference unit (MCU) based call-in conference, each participant uses a separate call to invite himself to the MCU.

Call leg: A call leg is identified by the combination of Call-ID, To and From.

Client: An application program that sends SIP requests. Clients may or may not interact directly with a human user. User agents and proxies contain clients (and servers).

Conference: A multimedia session (see below), identified by a common session description. A conference can have zero or more members and includes the cases of a multicast conference, a full-mesh conference and a two-party "telephone call", as well as combinations of these. Any number of calls can be used to create a conference.

Downstream: Requests sent in the direction from the caller to the callee (i.e., user agent client to user agent server).

Final response: A response that terminates a SIP transaction, as opposed to a provisional response that does not. All 2xx, 3xx, 4xx, 5xx and 6xx responses are final.

Initiator, calling party, caller: The party initiating a conference invitation. Note that the calling party does not have to be the same as the one creating the conference.

Invitation: A request sent to a user (or service) requesting participation in a session. A successful SIP invitation consists of two transactions: an INVITE request followed by an ACK request.

Invitee, invited user, called party, callee: The person or service that the calling party is trying to invite to a conference.

Isomorphic request or response: Two requests or responses are defined to be isomorphic for the purposes of this document if they have the same values for the Call-ID, To, From and CSeq header fields. In addition, isomorphic requests have to have the same Request-URI.

Location server: See location service.

Location service: A location service is used by a SIP redirect or proxy server to obtain information about a callee's possible location(s). Location services are offered by location servers. Location servers MAY be co-located with a SIP server, but the manner in which a SIP server requests location services is beyond the scope of this document.

Parallel search: In a parallel search, a proxy issues several requests to possible user locations upon receiving an incoming request. Rather than issuing one request and then waiting for the final response before issuing the next request as in a sequential search , a parallel search issues requests without waiting for the result of previous requests.

Provisional response: A response used by the server to indicate progress, but that does not terminate a SIP transaction. 1xx responses are provisional, other responses are considered final.

Proxy, proxy server: An intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, possibly after translation, to other servers. A proxy interprets, and, if necessary, rewrites a request message before forwarding it.

Redirect server: A redirect server is a server that accepts a SIP request, maps the address into zero or more new addresses and returns these addresses to the client. Unlike a proxy server , it does not initiate its own SIP request. Unlike a user agent server , it does not accept calls.

Registrar: A registrar is a server that accepts REGISTER requests. A registrar is typically co-located with a proxy or redirect server and MAY offer location services.

Ringback: Ringback is the signaling tone produced by the calling client's application indicating that a called party is being alerted (ringing).

Server: A server is an application program that accepts requests in order to service requests and sends back responses to those requests. Servers are either proxy, redirect or user agent servers or registrars.

Session: From the SDP specification: "A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session." (RFC 2327 [6]) (A session as defined for SDP can comprise one or more RTP sessions.) As defined, a callee can be invited several times, by different calls, to the same session. If SDP is used, a session is defined by the concatenation of the user name , session id , network type , address type and address elements in the origin field.

(SIP) transaction: A SIP transaction occurs between a client and a server and comprises all messages from the first request sent from the client to the server up to a final (non-1xx) response sent from the server to the client. A transaction is identified by the CSeq sequence number (Section 6.17) within a single call leg. The ACK request has the same CSeq number as the corresponding INVITE request, but comprises a transaction of its own.

Upstream: Responses sent in the direction from the user agent server to the user agent client.

URL-encoded: A character string encoded according to RFC 1738, Section 2.2 [13].

User agent client (UAC), calling user agent: A user agent client is a client application that initiates the SIP request.

User agent server (UAS), called user agent: A user agent server is a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user. The response accepts, rejects or redirects the request.

User agent (UA): An application which contains both a user agent client and user agent server.

An application program MAY be capable of acting both as a client and a server. For example, a typical multimedia conference control application would act as a user agent client to initiate calls or to

invite others to conferences and as a user agent server to accept invitations. The properties of the different SIP server types are summarized in Table 1.

property	redirect server	proxy server	user agent server	registrar
also acts as a SIP client	no	yes	no	no
returns 1xx status	yes	yes	yes	yes
returns 2xx status	no	yes	yes	yes
returns 3xx status	yes	yes	yes	yes
returns 4xx status	yes	yes	yes	yes
returns 5xx status	yes	yes	yes	yes
returns 6xx status	no	yes	yes	yes
inserts Via header	no	yes	no	no
accepts ACK	yes	yes	yes	no

Table 1: Properties of the different SIP server types

1.4 Overview of SIP Operation

This section explains the basic protocol functionality and operation. Callers and callees are identified by SIP addresses, described in Section 1.4.1. When making a SIP call, a caller first locates the appropriate server (Section 1.4.2) and then sends a SIP request (Section 1.4.3). The most common SIP operation is the invitation (Section 1.4.4). Instead of directly reaching the intended callee, a SIP request may be redirected or may trigger a chain of new SIP requests by proxies (Section 1.4.5). Users can register their location(s) with SIP servers (Section 4.2.6).

1.4.1 SIP Addressing

The "objects" addressed by SIP are users at hosts, identified by a SIP URL. The SIP URL takes a form similar to a mailto or telnet URL, i.e., user@host. The user part is a user name or a telephone number. The host part is either a domain name or a numeric network address. See section 2 for a detailed discussion of SIP URL's.

A user's SIP address can be obtained out-of-band, can be learned via existing media agents, can be included in some mailers' message headers, or can be recorded during previous invitation interactions. In many cases, a user's SIP URL can be guessed from their email address.

A SIP URL address can designate an individual (possibly located at one of several end systems), the first available person from a group of individuals or a whole group. The form of the address, for example, sip:sales@example.com , is not sufficient, in general, to determine the intent of the caller.

If a user or service chooses to be reachable at an address that is guessable from the person's name and organizational affiliation, the traditional method of ensuring privacy by having an unlisted "phone" number is compromised. However, unlike traditional telephony, SIP offers authentication and access control mechanisms and can avail itself of lower-layer security mechanisms, so that client software can reject unauthorized or undesired call attempts.

1.4.2 Locating a SIP Server

When a client wishes to send a request, the client either sends it to a locally configured SIP proxy server (as in HTTP), independent of the Request-URI, or sends it to the IP address and port corresponding to the Request-URI.

For the latter case, the client must determine the protocol, port and IP address of a server to which to send the request. A client SHOULD follow the steps below to obtain this information, but MAY follow the alternative, optional procedure defined in Appendix D. At each step, unless stated otherwise, the client SHOULD try to contact a server at the port number listed in the Request-URI. If no port number is present in the Request-URI, the client uses port 5060. If the Request-URI specifies a protocol (TCP or UDP), the client contacts the server using that protocol. If no protocol is specified, the client tries UDP (if UDP is supported). If the attempt fails, or if the client doesn't support UDP but supports TCP, it then tries TCP.

A client SHOULD be able to interpret explicit network notifications (such as ICMP messages) which indicate that a server is not reachable, rather than relying solely on timeouts. (For socket-based programs: For TCP, connect() returns ECONNREFUSED if the client could not connect to a server at that address. For UDP, the socket needs to be bound to the destination address using connect() rather than sendto() or similar so that a second write() fails with ECONNREFUSED if there is no server listening) If the client finds the server is not reachable at a particular address, it SHOULD behave as if it had received a 400-class error response to that request.

The client tries to find one or more addresses for the SIP server by querying DNS. The procedure is as follows:

1. If the host portion of the Request-URI is an IP address, the client contacts the server at the given address. Otherwise, the client proceeds to the next step.

2. The client queries the DNS server for address records for the host portion of the Request-URI. If the DNS server returns no address records, the client stops, as it has been unable to locate a server. By address record, we mean A RR's, AAAA RR's, or other similar address records, chosen according to the client's network protocol capabilities.

There are no mandatory rules on how to select a host name for a SIP server. Users are encouraged to name their SIP servers using the sip.domainname (i.e., sip.example.com) convention, as specified in RFC 2219 [16]. Users may only know an email address instead of a full SIP URL for a callee, however. In that case, implementations may be able to increase the likelihood of reaching a SIP server for that domain by constructing a SIP URL from that email address by prefixing the host name with "sip.". In the future, this mechanism is likely to become unnecessary as better DNS techniques, such as the one in Appendix D, become widely available.

A client MAY cache a successful DNS query result. A successful query is one which contained records in the answer, and a server was contacted at one of the addresses from the answer. When the client wishes to send a request to the same host, it MUST start the search as if it had just received this answer from the name server. The client MUST follow the procedures in RFC1035 [15] regarding DNS cache invalidation when the DNS time-to-live expires.

1.4.3 SIP Transaction

Once the host part has been resolved to a SIP server, the client sends one or more SIP requests to that server and receives one or more responses from the server. A request (and its retransmissions) together with the responses triggered by that request make up a SIP transaction. All responses to a request contain the same values in the Call-ID, CSeq, To, and From fields (with the possible addition of a tag in the To field (section 6.37)). This allows responses to be matched with requests. The ACK request following an INVITE is not part of the transaction since it may traverse a different set of hosts.

If TCP is used, request and responses within a single SIP transaction are carried over the same TCP connection (see Section 10). Several SIP requests from the same client to the same server MAY use the same TCP connection or MAY use a new connection for each request.

If the client sent the request via unicast UDP, the response is sent to the address contained in the next Via header field (Section 6.40) of the response. If the request is sent via multicast UDP, the response is directed to the same multicast address and destination port. For UDP, reliability is achieved using retransmission (Section 10).

The SIP message format and operation is independent of the transport protocol.

1.4.4 SIP Invitation

A successful SIP invitation consists of two requests, INVITE followed by ACK. The INVITE (Section 4.2.1) request asks the callee to join a particular conference or establish a two-party conversation. After the callee has agreed to participate in the call, the caller confirms that it has received that response by sending an ACK (Section 4.2.2) request. If the caller no longer wants to participate in the call, it sends a BYE request instead of an ACK.

The INVITE request typically contains a session description, for example written in SDP (RFC 2327 [6]) format, that provides the called party with enough information to join the session. For multicast sessions, the session description enumerates the media types and formats that are allowed to be distributed to that session. For a unicast session, the session description enumerates the media types and formats that the caller is willing to use and where it wishes the media data to be sent. In either case, if the callee wishes to accept the call, it responds to the invitation by returning a similar description listing the media it wishes to use. For a multicast session, the callee SHOULD only return a session description if it is unable to receive the media indicated in the caller's description or wants to receive data via unicast.

The protocol exchanges for the INVITE method are shown in Fig. 1 for a proxy server and in Fig. 2 for a redirect server. (Note that the messages shown in the figures have been abbreviated slightly.) In Fig. 1, the proxy server accepts the INVITE request (step 1), contacts the location service with all or parts of the address (step 2) and obtains a more precise location (step 3). The proxy server then issues a SIP INVITE request to the address(es) returned by the location service (step 4). The user agent server alerts the user (step 5) and returns a success indication to the proxy server (step

6). The proxy server then returns the success result to the original caller (step 7). The receipt of this message is confirmed by the caller using an ACK request, which is forwarded to the callee (steps 8 and 9). Note that an ACK can also be sent directly to the callee, bypassing the proxy. All requests and responses have the same Call-ID.

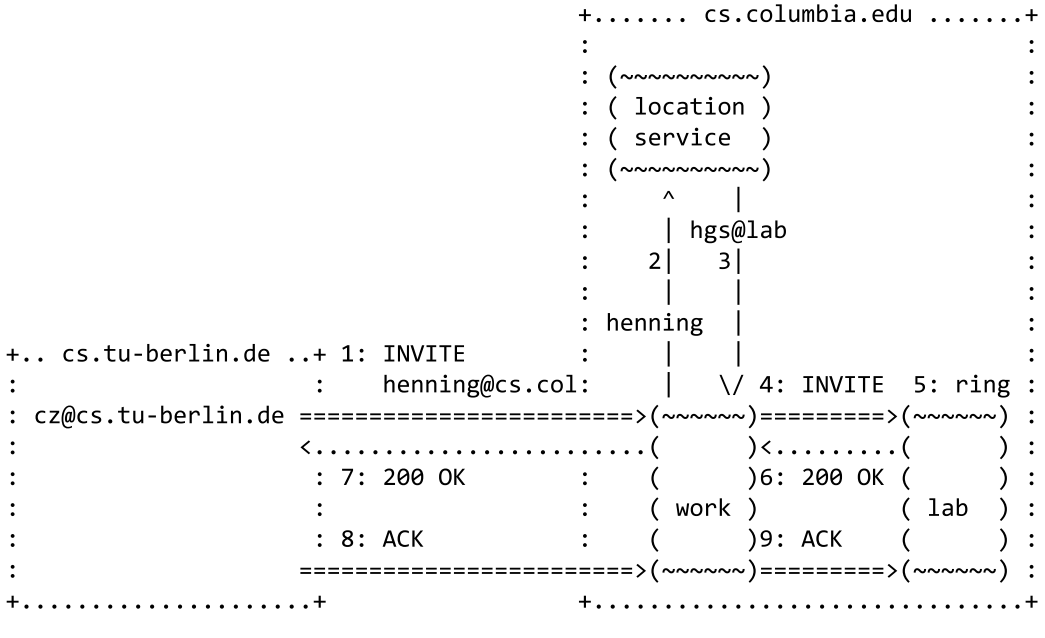


Figure 1: Example of SIP proxy server

The redirect server shown in Fig. 2 accepts the INVITE request (step 1), contacts the location service as before (steps 2 and 3) and, instead of contacting the newly found address itself, returns the address to the caller (step 4), which is then acknowledged via an ACK

request (step 5). The caller issues a new request, with the same call-ID but a higher CSeq, to the address returned by the first server (step 6). In the example, the call succeeds (step 7). The caller and callee complete the handshake with an ACK (step 8).

The next section discusses what happens if the location service returns more than one possible alternative.

1.4.5 Locating a User

A callee may move between a number of different end systems over time. These locations can be dynamically registered with the SIP server (Sections 1.4.7, 4.2.6). A location server MAY also use one or more other protocols, such as finger (RFC 1288 [17]), rwhois (RFC 2167 [18]), LDAP (RFC 1777 [19]), multicast-based protocols [20] or operating-system dependent mechanisms to actively determine the end system where a user might be reachable. A location server MAY return several locations because the user is logged in at several hosts simultaneously or because the location server has (temporarily) inaccurate information. The SIP server combines the results to yield a list of a zero or more locations.

The action taken on receiving a list of locations varies with the type of SIP server. A SIP redirect server returns the list to the client as Contact headers (Section 6.13). A SIP proxy server can sequentially or in parallel try the addresses until the call is successful (2xx response) or the callee has declined the call (6xx response). With sequential attempts, a proxy server can implement an "anycast" service.

If a proxy server forwards a SIP request, it MUST add itself to the beginning of the list of forwarders noted in the Via (Section 6.40) headers. The Via trace ensures that replies can take the same path back, ensuring correct operation through compliant firewalls and avoiding request loops. On the response path, each host MUST remove its Via, so that routing internal information is hidden from the callee and outside networks. A proxy server MUST check that it does not generate a request to a host listed in the Via sent-by, via-received or via-maddr parameters (Section 6.40). (Note: If a host has several names or network addresses, this does not always work. Thus, each host also checks if it is part of the Via list.)

A SIP invitation may traverse more than one SIP proxy server. If one of these "forks" the request, i.e., issues more than one request in response to receiving the invitation request, it is possible that a client is reached, independently, by more than one copy of the

invitation request. Each of these copies bears the same Call-ID. The user agent MUST return the same status response returned in the first response. Duplicate requests are not an error.

1.4.6 Changing an Existing Session

In some circumstances, it is desirable to change the parameters of an existing session. This is done by re-issuing the INVITE, using the same Call-ID, but a new or different body or header fields to convey the new information. This re INVITE MUST have a higher CSeq than any previous request from the client to the server.

For example, two parties may have been conversing and then want to add a third party, switching to multicast for efficiency. One of the participants invites the third party with the new multicast address and simultaneously sends an INVITE to the second party, with the new multicast session description, but with the old call identifier.

1.4.7 Registration Services

The REGISTER request allows a client to let a proxy or redirect server know at which address(es) it can be reached. A client MAY also use it to install call handling features at the server.

1.5 Protocol Properties

1.5.1 Minimal State

A single conference session or call involves one or more SIP request-response transactions. Proxy servers do not have to keep state for a particular call, however, they MAY maintain state for a single SIP transaction, as discussed in Section 12. For efficiency, a server MAY cache the results of location service requests.

1.5.2 Lower-Layer-Protocol Neutral

SIP makes minimal assumptions about the underlying transport and network-layer protocols. The lower-layer can provide either a packet or a byte stream service, with reliable or unreliable service.

In an Internet context, SIP is able to utilize both UDP and TCP as transport protocols, among others. UDP allows the application to more carefully control the timing of messages and their retransmission, to perform parallel searches without requiring TCP connection state for each outstanding request, and to use multicast. Routers can more readily snoop SIP UDP packets. TCP allows easier passage through existing firewalls.

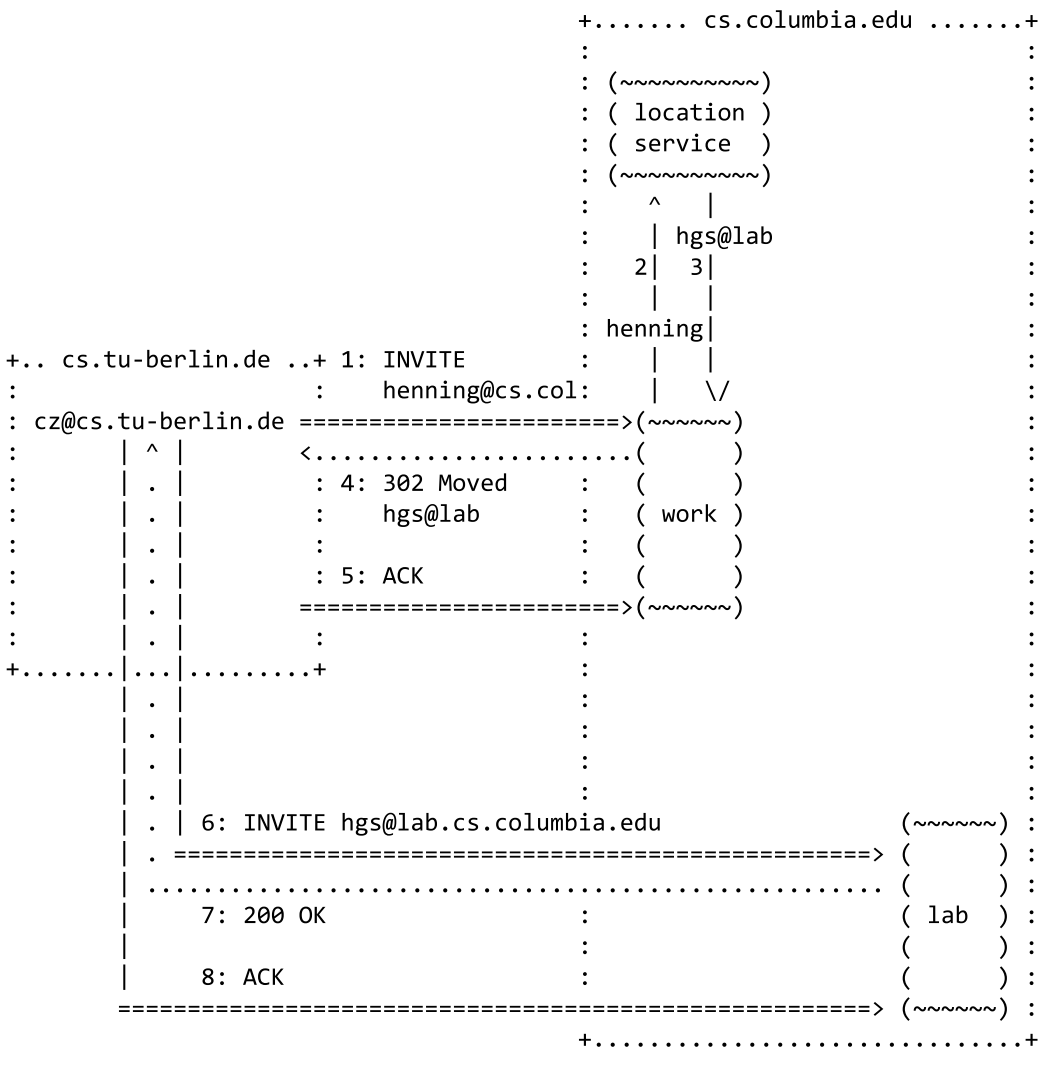


Figure 2: Example of SIP redirect server

When TCP is used, SIP can use one or more connections to attempt to contact a user or to modify parameters of an existing conference. Different SIP requests for the same SIP call MAY use different TCP connections or a single persistent connection, as appropriate.

For concreteness, this document will only refer to Internet protocols. However, SIP MAY also be used directly with protocols such as ATM AAL5, IPX, frame relay or X.25. The necessary naming conventions are beyond the scope of this document. User agents SHOULD implement both UDP and TCP transport. Proxy, registrar, and redirect servers MUST implement both UDP and TCP transport.

1.5.3 Text-Based

SIP is text-based, using ISO 10646 in UTF-8 encoding throughout. This allows easy implementation in languages such as Java, Tcl and Perl, allows easy debugging, and most importantly, makes SIP flexible and extensible. As SIP is used for initiating multimedia conferences rather than delivering media data, it is believed that the additional overhead of using a text-based protocol is not significant.

2 SIP Uniform Resource Locators

SIP URLs are used within SIP messages to indicate the originator (From), current destination (Request-URI) and final recipient (To) of a SIP request, and to specify redirection addresses (Contact). A SIP URL can also be embedded in web pages or other hyperlinks to indicate that a particular user or service can be called via SIP. When used as a hyperlink, the SIP URL indicates the use of the INVITE method.

The SIP URL scheme is defined to allow setting SIP request-header fields and the SIP message-body.

> This corresponds to the use of mailto: URLs. It makes it possible, for example, to specify the subject, urgency or media types of calls initiated through a web page or as part of an email message.

A SIP URL follows the guidelines of RFC 2396 [12] and has the syntax shown in Fig. 3. The syntax is described using Augmented Backus-Naur Form (See Section C). Note that reserved characters have to be escaped and that the "set of characters reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding" [12].

```
SIP-URL         = "sip:" [ userinfo "@" ] hostport
                  url-parameters [ headers ]
userinfo        = user [ ":" password ]
user            = *( unreserved | escaped
                | "&" | "=" | "+" | "$" | "," )
password        = *( unreserved | escaped
                | "&" | "=" | "+" | "$" | "," )
hostport        = host [ ":" port ]
host            = hostname | IPv4address
hostname        = *( domainlabel "." ) toplabel [ "." ]
domainlabel     = alphanum | alphanum *( alphanum | "-" ) alphanum
toplabel        = alpha | alpha *( alphanum | "-" ) alphanum
IPv4address     = 1*digit "." 1*digit "." 1*digit "." 1*digit
port            = *digit
url-parameters  = *( ";" url-parameter )
url-parameter   = transport-param | user-param | method-param
                | ttl-param | maddr-param | other-param
transport-param = "transport=" ( "udp" | "tcp" )
ttl-param       = "ttl=" ttl
ttl             = 1*3DIGIT       ; 0 to 255
maddr-param     = "maddr=" host
user-param      = "user=" ( "phone" | "ip" )
method-param    = "method=" Method
tag-param       = "tag=" UUID
UUID            = 1*( hex | "-" )
other-param     = ( token | ( token "=" ( token | quoted-string )))
headers         = "?" header *( "&" header )
header          = hname "=" hvalue
hname           = 1*uric
hvalue          = *uric
uric            = reserved | unreserved | escaped
reserved        = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                  "$" | ","
digits          = 1*DIGIT
```

Figure 3: SIP URL syntax

The URI character classes referenced above are described in Appendix C.

The components of the SIP URI have the following meanings.

```
telephone-subscriber  = global-phone-number | local-phone-number
   global-phone-number   = "+" 1*phonedigit [isdn-subaddress]
                             [post-dial]
   local-phone-number    = 1*(phonedigit | dtmf-digit |
                             pause-character) [isdn-subaddress]
                             [post-dial]
   isdn-subaddress       = ";isub=" 1*phonedigit
   post-dial             = ";postd=" 1*(phonedigit | dtmf-digit
                         |  pause-character)
   phonedigit            = DIGIT | visual-separator
   visual-separator      = "-" | "."
   pause-character       = one-second-pause | wait-for-dial-tone
   one-second-pause      = "p"
   wait-for-dial-tone    = "w"
   dtmf-digit            = "*" | "#" | "A" | "B" | "C" | "D"
```

Figure 4: SIP URL syntax; telephone subscriber

user: If the host is an Internet telephony gateway, the user field MAY also encode a telephone number using the notation of telephone-subscriber (Fig. 4). The telephone number is a special case of a user name and cannot be distinguished by a BNF. Thus, a URL parameter, user, is added to distinguish telephone numbers from user names. The phone identifier is to be used when connecting to a telephony gateway. Even without this parameter, recipients of SIP URLs MAY interpret the pre-@ part as a phone number if local restrictions on the name space for user name allow it.

password: The SIP scheme MAY use the format "user:password" in the userinfo field. The use of passwords in the userinfo is NOT RECOMMENDED, because the passing of authentication information in clear text (such as URIs) has proven to be a security risk in almost every case where it has been used.

host: The mailto: URL and RFC 822 email addresses require that numeric host addresses ("host numbers") are enclosed in square brackets (presumably, since host names might be numeric), while host numbers without brackets are used for all other URLs. The SIP URL requires the latter form, without brackets.

The issue of IPv6 literal addresses in URLs is being looked at elsewhere in the IETF. SIP implementers are advised to keep up to date on that activity.

port: The port number to send a request to. If not present, the procedures outlined in Section 1.4.2 are used to determine the port number to send a request to.

URL parameters: SIP URLs can define specific parameters of the request. URL parameters are added after the host component and are separated by semi-colons. The transport parameter determines the the transport mechanism (UDP or TCP). UDP is to be assumed when no explicit transport parameter is included. The maddr parameter provides the server address to be contacted for this user, overriding the address supplied in the host field. This address is typically a multicast address, but could also be the address of a backup server. The ttl parameter determines the time-to-live value of the UDP multicast packet and MUST only be used if maddr is a multicast address and the transport protocol is UDP. The user parameter was described above. For example, to specify to call j.doe@big.com using multicast to 239.255.255.1 with a ttl of 15, the following URL would be used:

```
sip:j.doe@big.com;maddr=239.255.255.1;ttl=15
```

The transport, maddr, and ttl parameters MUST NOT be used in the From and To header fields and the Request-URI; they are ignored if present.

Headers: Headers of the SIP request can be defined with the "?" mechanism within a SIP URL. The special hname "body" indicates that the associated hvalue is the message-body of the SIP INVITE request. Headers MUST NOT be used in the From and To header fields and the Request-URI; they are ignored if present. hname and hvalue are encodings of a SIP header name and value, respectively. All URL reserved characters in the header names and values MUST be escaped.

Method: The method of the SIP request can be specified with the method parameter. This parameter MUST NOT be used in the From and To header fields and the Request-URI; they are ignored if present.

Table 2 summarizes where the components of the SIP URL can be used and what default values they assume if not present.

Examples of SIP URLs are:

	default	Req.-URI	To	From	Contact	external
user	--	x	x	x	x	x
password	--	x	x		x	x
host	mandatory	x	x	x	x	x
port	5060	x	x	x	x	x
user-param	ip	x	x	x	x	x
method	INVITE				x	x
maddr-param	--				x	x
ttl-param	1				x	x
transp.-param	--				x	x
headers	--				x	x

Table 2: Use and default values of URL components for SIP headers, Request-URI and references

```
sip:j.doe@big.com
sip:j.doe:secret@big.com;transport=tcp
sip:j.doe@big.com?subject=project
sip:+1-212-555-1212:1234@gateway.com;user=phone
sip:1212@gateway.com
sip:alice@10.1.2.3
sip:alice@example.com
sip:alice%40example.com@gateway.com
sip:alice@registrar.com;method=REGISTER
```

Within a SIP message, URLs are used to indicate the source and intended destination of a request, redirection addresses and the current destination of a request. Normally all these fields will contain SIP URLs.

SIP URLs are case-insensitive, so that for example the two URLs sip:j.doe@example.com and SIP:J.Doe@Example.com are equivalent. All URL parameters are included when comparing SIP URLs for equality.

SIP header fields MAY contain non-SIP URLs. As an example, if a call from a telephone is relayed to the Internet via SIP, the SIP From header field might contain a phone URL.

3 SIP Message Overview

SIP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279 [21]). Senders MUST terminate lines with a CRLF, but receivers MUST also interpret CR and LF by themselves as line terminators.

Except for the above difference in character sets, much of the message syntax is and header fields are identical to HTTP/1.1; rather than repeating the syntax and semantics here we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification (RFC 2068 [11]). In addition, we describe SIP in both prose and an augmented Backus-Naur form (ABNF). See section C for an overview of ABNF.

Note, however, that SIP is not an extension of HTTP.

Unlike HTTP, SIP MAY use UDP. When sent over TCP or UDP, multiple SIP transactions can be carried in a single TCP connection or UDP datagram. UDP datagrams, including all headers, SHOULD NOT be larger than the path maximum transmission unit (MTU) if the MTU is known, or 1500 bytes if the MTU is unknown.

The 1500 bytes accommodates encapsulation within the "typical" ethernet MTU without IP fragmentation. Recent studies [22] indicate that an MTU of 1500 bytes is a reasonable assumption. The next lower common MTU values are 1006 bytes for SLIP and 296 for low-delay PPP (RFC 1191 [23]). Thus, another reasonable value would be a message size of 950 bytes, to accommodate packet headers within the SLIP MTU without fragmentation.

A SIP message is either a request from a client to a server, or a response from a server to a client.

```
SIP-message  =  Request | Response
```

Both Request (section 4) and Response (section 5) messages use the generic-message format of RFC 822 [24] for transferring entities (the body of the message). Both types of messages consist of a start-line, one or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the carriage-return line-feed (CRLF)) indicating the end of the header fields, and an optional message-body. To avoid confusion with similar-named headers in HTTP, we refer to the headers describing the message body as entity headers. These components are described in detail in the upcoming sections.

```
generic-message  =  start-line
                    *message-header
```

```
                        CRLF
                        [ message-body ]

      start-line       =  Request-Line |     ;Section 4.1
                          Status-Line        ;Section 5.1



      message-header   =  ( general-header
                          | request-header
                          | response-header
                          | entity-header )
```

In the interest of robustness, any leading empty line(s) MUST be ignored. In other words, if the Request or Response message begins with one or more CRLF, CR, or LFs, these characters MUST be ignored.

4 Request

The Request message format is shown below:

```
     Request  =  Request-Line       ;  Section 4.1
                 *( general-header
                 | request-header
                 | entity-header )
                 CRLF
                 [ message-body ]   ;  Section 8
```

4.1 Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF are allowed except in the final CRLF sequence.

```
     Request-Line  =  Method SP Request-URI SP SIP-Version CRLF
```

```
general-header   =  Accept               ; Section 6.7
                 |  Accept-Encoding      ; Section 6.8
                 |  Accept-Language      ; Section 6.9
                 |  Call-ID              ; Section 6.12
                 |  Contact              ; Section 6.13
                 |  CSeq                 ; Section 6.17
                 |  Date                 ; Section 6.18
                 |  Encryption           ; Section 6.19
                 |  Expires              ; Section 6.20
                 |  From                 ; Section 6.21
                 |  Record-Route         ; Section 6.29
                 |  Timestamp            ; Section 6.36
                 |  To                   ; Section 6.37
                 |  Via                  ; Section 6.40
entity-header    =  Content-Encoding     ; Section 6.14
                 |  Content-Length       ; Section 6.15
                 |  Content-Type         ; Section 6.16
request-header   =  Authorization        ; Section 6.11
                 |  Contact              ; Section 6.13
                 |  Hide                 ; Section 6.22
                 |  Max-Forwards         ; Section 6.23
                 |  Organization         ; Section 6.24
                 |  Priority             ; Section 6.25
                 |  Proxy-Authorization  ; Section 6.27
                 |  Proxy-Require        ; Section 6.28
                 |  Route                ; Section 6.33
                 |  Require              ; Section 6.30
                 |  Response-Key         ; Section 6.31
                 |  Subject              ; Section 6.35
                 |  User-Agent           ; Section 6.39
response-header  =  Allow                ; Section 6.10
                 |  Proxy-Authenticate   ; Section 6.26
                 |  Retry-After          ; Section 6.32
                 |  Server               ; Section 6.34
                 |  Unsupported          ; Section 6.38
                 |  Warning              ; Section 6.41
                 |  WWW-Authenticate     ; Section 6.42
```

Table 3: SIP headers

4.2 Methods

The methods are defined below. Methods that are not supported by a proxy or redirect server are treated by that server as if they were an OPTIONS method and forwarded accordingly. Methods that are not

supported by a user agent server or registrar cause a 501 (Not Implemented) response to be returned (Section 7). As in HTTP, the Method token is case-sensitive.

```
Method  =  "INVITE" | "ACK" | "OPTIONS" | "BYE"
           | "CANCEL" | "REGISTER"
```

4.2.1 INVITE

The INVITE method indicates that the user or service is being invited to participate in a session. The message body contains a description of the session to which the callee is being invited. For two-party calls, the caller indicates the type of media it is able to receive and possibly the media it is willing to send as well as their parameters such as network destination. A success response MUST indicate in its message body which media the callee wishes to receive and MAY indicate the media the callee is going to send.

> Not all session description formats have the ability to indicate sending media.

A server MAY automatically respond to an invitation for a conference the user is already participating in, identified either by the SIP Call-ID or a globally unique identifier within the session description, with a 200 (OK) response.

If a user agent receives an INVITE request for an existing call leg with a higher CSeq sequence number than any previous INVITE for the same Call-ID, it MUST check any version identifiers in the session description or, if there are no version identifiers, the content of the session description to see if it has changed. It MUST also inspect any other header fields for changes. If there is a change, the user agent MUST update any internal state or information generated as a result of that header. If the session description has changed, the user agent server MUST adjust the session parameters accordingly, possibly after asking the user for confirmation. (Versioning of the session description can be used to accommodate the capabilities of new arrivals to a conference, add or delete media or change from a unicast to a multicast conference.)

This method MUST be supported by SIP proxy, redirect and user agent servers as well as clients.

4.2.2 ACK

The ACK request confirms that the client has received a final response to an INVITE request. (ACK is used only with INVITE requests.) 2xx responses are acknowledged by client user agents, all other final responses by the first proxy or client user agent to receive the response. The Via is always initialized to the host that originates the ACK request, i.e., the client user agent after a 2xx response or the first proxy to receive a non-2xx final response. The ACK request is forwarded as the corresponding INVITE request, based on its Request-URI. See Section 10 for details.

The ACK request MAY contain a message body with the final session description to be used by the callee. If the ACK message body is empty, the callee uses the session description in the INVITE request.

A proxy server receiving an ACK request after having sent a 3xx, 4xx, 5xx, or 6xx response must make a determination about whether the ACK is for it, or for some user agent or proxy server further downstream. This determination is made by examining the tag in the To field. If the tag in the ACK To header field matches the tag in the To header field of the response, and the From, CSeq and Call-ID header fields in the response match those in the ACK, the ACK is meant for the proxy server. Otherwise, the ACK SHOULD be proxied downstream as any other request.

> It is possible for a user agent client or proxy server to receive multiple 3xx, 4xx, 5xx, and 6xx responses to a request along a single branch. This can happen under various error conditions, typically when a forking proxy transitions from stateful to stateless before receiving all responses. The various responses will all be identical, except for the tag in the To field, which is different for each one. It can therefore be used as a means to disambiguate them.

This method MUST be supported by SIP proxy, redirect and user agent servers as well as clients.

4.2.3 OPTIONS

The server is being queried as to its capabilities. A server that believes it can contact the user, such as a user agent where the user is logged in and has been recently active, MAY respond to this request with a capability set. A called user agent MAY return a status reflecting how it would have responded to an invitation, e.g.,

600 (Busy). Such a server SHOULD return an Allow header field indicating the methods that it supports. Proxy and redirect servers simply forward the request without indicating their capabilities.

This method MUST be supported by SIP proxy, redirect and user agent servers, registrars and clients.

4.2.4 BYE

The user agent client uses BYE to indicate to the server that it wishes to release the call. A BYE request is forwarded like an INVITE request and MAY be issued by either caller or callee. A party to a call SHOULD issue a BYE request before releasing a call ("hanging up"). A party receiving a BYE request MUST cease transmitting media streams specifically directed at the party issuing the BYE request.

If the INVITE request contained a Contact header, the callee SHOULD send a BYE request to that address rather than the From address.

This method MUST be supported by proxy servers and SHOULD be supported by redirect and user agent SIP servers.

4.2.5 CANCEL

The CANCEL request cancels a pending request with the same Call-ID, To, From and CSeq (sequence number only) header field values, but does not affect a completed request. (A request is considered completed if the server has returned a final status response.)

A user agent client or proxy client MAY issue a CANCEL request at any time. A proxy, in particular, MAY choose to send a CANCEL to destinations that have not yet returned a final response after it has received a 2xx or 6xx response for one or more of the parallel-search requests. A proxy that receives a CANCEL request forwards the request to all destinations with pending requests.

The Call-ID, To, the numeric part of CSeq and From headers in the CANCEL request are identical to those in the original request. This allows a CANCEL request to be matched with the request it cancels. However, to allow the client to distinguish responses to the CANCEL from those to the original request, the CSeq Method component is set to CANCEL. The Via header field is initialized to the proxy issuing the CANCEL request. (Thus, responses to this CANCEL request only reach the issuing proxy.)

Once a user agent server has received a CANCEL, it MUST NOT issue a 2xx response for the cancelled original request.

A redirect or user agent server receiving a CANCEL request responds with a status of 200 (OK) if the transaction exists and a status of 481 (Transaction Does Not Exist) if not, but takes no further action. In particular, any existing call is unaffected.

> The BYE request cannot be used to cancel branches of a parallel search, since several branches may, through intermediate proxies, find the same user agent server and then terminate the call. To terminate a call instead of just pending searches, the UAC must use BYE instead of or in addition to CANCEL. While CANCEL can terminate any pending request other than ACK or CANCEL, it is typically useful only for INVITE. 200 responses to INVITE and 200 responses to CANCEL are distinguished by the method in the Cseq header field, so there is no ambiguity.

This method MUST be supported by proxy servers and SHOULD be supported by all other SIP server types.

4.2.6 REGISTER

A client uses the REGISTER method to register the address listed in the To header field with a SIP server.

A user agent MAY register with a local server on startup by sending a REGISTER request to the well-known "all SIP servers" multicast address "sip.mcast.net" (224.0.1.75). This request SHOULD be scoped to ensure it is not forwarded beyond the boundaries of the administrative system. This MAY be done with either TTL or administrative scopes [25], depending on what is implemented in the network. SIP user agents MAY listen to that address and use it to become aware of the location of other local users [20]; however, they do not respond to the request. A user agent MAY also be configured with the address of a registrar server to which it sends a REGISTER request upon startup.

Requests are processed in the order received. Clients SHOULD avoid sending a new registration (as opposed to a retransmission) until they have received the response from the server for the previous one.

> Clients may register from different locations, by necessity using different Call-ID values. Thus, the CSeq value cannot be used to enforce ordering. Since registrations are additive, ordering is less of a problem than if each REGISTER request completely replaced all earlier ones.

The meaning of the REGISTER request-header fields is defined as follows. We define "address-of-record" as the SIP address that the registry knows the registrand, typically of the form "user@domain" rather than "user@host". In third-party registration, the entity issuing the request is different from the entity being registered.

To: The To header field contains the address-of-record whose registration is to be created or updated.

From: The From header field contains the address-of-record of the person responsible for the registration. For first-party registration, it is identical to the To header field value.

Request-URI: The Request-URI names the destination of the registration request, i.e., the domain of the registrar. The user name MUST be empty. Generally, the domains in the Request-URI and the To header field have the same value; however, it is possible to register as a "visitor", while maintaining one's name. For example, a traveler sip:alice@acme.com (To) might register under the Request-URI sip:atlanta.hiayh.org , with the former as the To header field and the latter as the Request-URI. The REGISTER request is no longer forwarded once it has reached the server whose authoritative domain is the one listed in the Request-URI.

Call-ID: All registrations from a client SHOULD use the same Call-ID header value, at least within the same reboot cycle.

Cseq: Registrations with the same Call-ID MUST have increasing CSeq header values. However, the server does not reject out-of-order requests.

Contact: The request MAY contain a Contact header field; future non-REGISTER requests for the URI given in the To header field SHOULD be directed to the address(es) given in the Contact header.

If the request does not contain a Contact header, the registration remains unchanged.

This is useful to obtain the current list of registrations in the response. Registrations using SIP URIs that differ in one or more of host, port, transport-param or maddr-param (see Figure 3) from an existing registration are added to the list of registrations. Other URI types are compared according to the standard URI equivalency rules for the URI schema. If the URIs are equivalent to that of an existing registration, the new registration replaces the

old one if it has a higher q value or, for the same value of q, if the ttl value is higher. All current registrations MUST share the same action value. Registrations that have a different action than current registrations for the same user MUST be rejected with status of 409 (Conflict).

A proxy server ignores the q parameter when processing non-REGISTER requests, while a redirect server simply returns that parameter in its Contact response header field.

Having the proxy server interpret the q parameter is not sufficient to guide proxy behavior, as it is not clear, for example, how long it is supposed to wait between trying addresses.

If the registration is changed while a user agent or proxy server processes an invitation, the new information SHOULD be used.

This allows a service known as "directed pick-up". In the telephone network, directed pickup permits a user at a remote station who hears his own phone ringing to pick up at that station, dial an access code, and be connected to the calling user as if he had answered his own phone.

A server MAY choose any duration for the registration lifetime. Registrations not refreshed after this amount of time SHOULD be silently discarded. Responses to a registration SHOULD include an Expires header (Section 6.20) or expires Contact parameters (Section 6.13), indicating the time at which the server will drop the registration. If none is present, one hour is assumed. Clients MAY request a registration lifetime by indicating the time in an Expires header in the request. A server SHOULD NOT use a higher lifetime than the one requested, but MAY use a lower one. A single address (if host-independent) MAY be registered from several different clients.

A client cancels an existing registration by sending a REGISTER request with an expiration time (Expires) of zero seconds for a particular Contact or the wildcard Contact designated by a "*" for all registrations. Registrations are matched based on the user, host, port and maddr parameters.

The server SHOULD return the current list of registrations in the 200 response as Contact header fields.

It is particularly important that REGISTER requests are authenticated since they allow to redirect future requests (see Section 13.2).

Beyond its use as a simple location service, this method is needed if there are several SIP servers on a single host. In that case, only one of the servers can use the default port number.

Support of this method is RECOMMENDED.

4.3 Request-URI

The Request-URI is a SIP URL as described in Section 2 or a general URI. It indicates the user or service to which this request is being addressed. Unlike the To field, the Request-URI MAY be re-written by proxies.

When used as a Request-URI, a SIP-URL MUST NOT contain the transport-param, maddr-param, ttl-param, or headers elements. A server that receives a SIP-URL with these elements removes them before further processing.

Typically, the UAC sets the Request-URI and To to the same SIP URL, presumed to remain unchanged over long time periods. However, if the UAC has cached a more direct path to the callee, e.g., from the Contact header field of a response to a previous request, the To would still contain the long-term, "public" address, while the Request-URI would be set to the cached address.

Proxy and redirect servers MAY use the information in the Request-URI and request header fields to handle the request and possibly rewrite the Request-URI. For example, a request addressed to the generic address sip:sales@acme.com is proxied to the particular person, e.g., sip:bob@ny.acme.com , with the To field remaining as sip:sales@acme.com. At ny.acme.com , Bob then designates Alice as the temporary substitute.

The host part of the Request-URI typically agrees with one of the host names of the receiving server. If it does not, the server SHOULD proxy the request to the address indicated or return a 404 (Not Found) response if it is unwilling or unable to do so. For example, the Request-URI and server host name can disagree in the case of a firewall proxy that handles outgoing calls. This mode of operation is similar to that of HTTP proxies.

If a SIP server receives a request with a URI indicating a scheme other than SIP which that server does not understand, the server MUST return a 400 (Bad Request) response. It MUST do this even if the To

header field contains a scheme it does understand. This is because proxies are responsible for processing the Request-URI; the To field is of end-to-end significance.

4.3.1 SIP Version

Both request and response messages include the version of SIP in use, and follow [H3.1] (with HTTP replaced by SIP, and HTTP/1.1 replaced by SIP/2.0) regarding version ordering, compliance requirements, and upgrading of version numbers. To be compliant with this specification, applications sending SIP messages MUST include a SIP-Version of "SIP/2.0".

4.4 Option Tags

Option tags are unique identifiers used to designate new options in SIP. These tags are used in Require (Section 6.30) and Unsupported (Section 6.38) fields.

Syntax:

```
option-tag  =  token
```

See Section C for a definition of token. The creator of a new SIP option MUST either prefix the option with their reverse domain name or register the new option with the Internet Assigned Numbers Authority (IANA). For example, "com.foo.mynewfeature" is an apt name for a feature whose inventor can be reached at "foo.com". Individual organizations are then responsible for ensuring that option names don't collide. Options registered with IANA have the prefix "org.iana.sip.", options described in RFCs have the prefix "org.ietf.rfc.N", where N is the RFC number. Option tags are case-insensitive.

4.4.1 Registering New Option Tags with IANA

When registering a new SIP option, the following information MUST be provided:

- Name and description of option. The name MAY be of any length, but SHOULD be no more than twenty characters long. The name MUST consist of alphanum (See Figure 3) characters only;

- Indication of who has change control over the option (for example, IETF, ISO, ITU-T, other international standardization bodies, a consortium or a particular company or group of companies);

- A reference to a further description, if available, for example (in order of preference) an RFC, a published paper, a patent filing, a technical report, documented source code or a computer manual;

- Contact information (postal and email address);

Registrations should be sent to iana@iana.org

This procedure has been borrowed from RTSP [4] and the RTP AVP [26].

5 Response

After receiving and interpreting a request message, the recipient responds with a SIP response message. The response message format is shown below:

```
Response  =  Status-Line        ;  Section 5.1
             *( general-header
             | response-header
             | entity-header )
             CRLF
             [ message-body ]   ;  Section 8
```

SIP's structure of responses is similar to [H6], but is defined explicitly here.

5.1 Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version (Section 4.3.1) followed by a numeric Status-Code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

```
Status-Line  =  SIP-version SP Status-Code SP Reason-Phrase CRLF
```

5.1.1 Status Codes and Reason Phrases

The Status-Code is a 3-digit integer result code that indicates the outcome of the attempt to understand and satisfy the request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata, whereas the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

```
     Status-Code     =  Informational                    ;Fig. 5
                     |  Success                          ;Fig. 5
                     |  Redirection                      ;Fig. 6
                     |  Client-Error                     ;Fig. 7
                     |  Server-Error                     ;Fig. 8
                     |  Global-Failure                   ;Fig. 9
                     |  extension-code
     extension-code  =  3DIGIT
     Reason-Phrase   =  *<TEXT-UTF8,  excluding CR, LF>
```

We provide an overview of the Status-Code below, and provide full definitions in Section 7. The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. SIP/2.0 allows 6 values for the first digit:

1xx: Informational -- request received, continuing to process the request;

2xx: Success -- the action was successfully received, understood, and accepted;

3xx: Redirection -- further action needs to be taken in order to complete the request;

4xx: Client Error -- the request contains bad syntax or cannot be fulfilled at this server;

5xx: Server Error -- the server failed to fulfill an apparently valid request;

6xx: Global Failure -- the request cannot be fulfilled at any server.

Figures 5 through 9 present the individual values of the numeric response codes, and an example set of corresponding reason phrases for SIP/2.0. These reason phrases are only recommended; they may be replaced by local equivalents without affecting the protocol. Note

that SIP adopts many HTTP/1.1 response codes. SIP/2.0 adds response codes in the range starting at x80 to avoid conflicts with newly defined HTTP response codes, and adds a new class, 6xx, of response codes.

SIP response codes are extensible. SIP applications are not required to understand the meaning of all registered response codes, though such understanding is obviously desirable. However, applications MUST understand the class of any response code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 response code of that class, with the exception that an unrecognized response MUST NOT be cached. For example, if a client receives an unrecognized response code of 431, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 (Bad Request) response code. In such cases, user agents SHOULD present to the user the message body returned with the response, since that message body is likely to include human-readable information which will explain the unusual status.

```
Informational  =  "100"  ;  Trying
               |  "180"  ;  Ringing
               |  "181"  ;  Call Is Being Forwarded
               |  "182"  ;  Queued
Success        =  "200"  ;  OK
```

Figure 5: Informational and success status codes

```
Redirection  =  "300"  ;  Multiple Choices
             |  "301"  ;  Moved Permanently
             |  "302"  ;  Moved Temporarily
             |  "303"  ;  See Other
             |  "305"  ;  Use Proxy
             |  "380"  ;  Alternative Service
```

Figure 6: Redirection status codes

```
Client-Error  =  "400"  ;  Bad Request
              |  "401"  ;  Unauthorized
              |  "402"  ;  Payment Required
              |  "403"  ;  Forbidden
              |  "404"  ;  Not Found
              |  "405"  ;  Method Not Allowed
              |  "406"  ;  Not Acceptable
              |  "407"  ;  Proxy Authentication Required
              |  "408"  ;  Request Timeout
              |  "409"  ;  Conflict
              |  "410"  ;  Gone
              |  "411"  ;  Length Required
              |  "413"  ;  Request Entity Too Large
              |  "414"  ;  Request-URI Too Large
              |  "415"  ;  Unsupported Media Type
              |  "420"  ;  Bad Extension
              |  "480"  ;  Temporarily not available
              |  "481"  ;  Call Leg/Transaction Does Not Exist
              |  "482"  ;  Loop Detected
              |  "483"  ;  Too Many Hops
              |  "484"  ;  Address Incomplete
              |  "485"  ;  Ambiguous
              |  "486"  ;  Busy Here
```

Figure 7: Client error status codes

```
Server-Error  =  "500"  ;  Internal Server Error
              |  "501"  ;  Not Implemented
              |  "502"  ;  Bad Gateway
              |  "503"  ;  Service Unavailable
              |  "504"  ;  Gateway Time-out
              |  "505"  ;  SIP Version not supported
```

Figure 8: Server error status codes

6 Header Field Definitions

SIP header fields are similar to HTTP header fields in both syntax and semantics. In particular, SIP header fields follow the syntax for message-header as described in [H4.2]. The rules for extending header fields over multiple lines, and use of multiple message-header fields with the same field-name, described in [H4.2] also apply to SIP. The

```
Global-Failure |  "600"  ;  Busy Everywhere
               |  "603"  ;  Decline
               |  "604"  ;  Does not exist anywhere
               |  "606"  ;  Not Acceptable
```

Figure 9: Global failure status codes

rules in [H4.2] regarding ordering of header fields apply to SIP, with the exception of Via fields, see below, whose order matters. Additionally, header fields which are hop-by-hop MUST appear before any header fields which are end-to-end. Proxies SHOULD NOT reorder header fields. Proxies add Via header fields and MAY add other hop-by-hop header fields. They can modify certain header fields, such as Max-Forwards (Section 6.23) and "fix up" the Via header fields with "received" parameters as described in Section 6.40.1. Proxies MUST NOT alter any fields that are authenticated (see Section 13.2).

The header fields required, optional and not applicable for each method are listed in Table 4 and Table 5. The table uses "o" to indicate optional, "m" mandatory and "-" for not applicable. A "*" indicates that the header fields are needed only if message body is not empty. See sections 6.15, 6.16 and 8 for details.

The "where" column describes the request and response types with which the header field can be used. "R" refers to header fields that can be used in requests (that is, request and general header fields). "r" designates a response or general-header field as applicable to all responses, while a list of numeric values indicates the status codes with which the header field can be used. "g" and "e" designate general (Section 6.1) and entity header (Section 6.2) fields, respectively. If a header field is marked "c", it is copied from the request to the response.

The "enc." column describes whether this message header field MAY be encrypted end-to-end. A "n" designates fields that MUST NOT be encrypted, while "c" designates fields that SHOULD be encrypted if encryption is used.

The "e-e" column has a value of "e" for end-to-end and a value of "h" for hop-by-hop header fields.

	where	enc.	e-e	ACK	BYE	CAN	INV	OPT	REG
Accept	R		e	-	-	-	o	o	o
Accept	415		e	-	-	-	o	o	o
Accept-Encoding	R		e	-	-	-	o	o	o
Accept-Encoding	415		e	-	-	-	o	o	o
Accept-Language	R		e	-	o	o	o	o	o
Accept-Language	415		e	-	o	o	o	o	o
Allow	200		e	-	-	-	-	m	-
Allow	405		e	o	o	o	o	o	o
Authorization	R		e	o	o	o	o	o	o
Call-ID	gc	n	e	m	m	m	m	m	m
Contact	R		e	o	-	-	o	o	o
Contact	1xx		e	-	-	-	o	o	-
Contact	2xx		e	-	-	-	o	o	o
Contact	3xx		e	-	o	-	o	o	o
Contact	485		e	-	o	-	o	o	o
Content-Encoding	e		e	o	-	-	o	o	o
Content-Length	e		e	o	-	-	o	o	o
Content-Type	e		e	*	-	-	*	*	*
CSeq	gc	n	e	m	m	m	m	m	m
Date	g		e	o	o	o	o	o	o
Encryption	g	n	e	o	o	o	o	o	o
Expires	g		e	-	-	-	o	-	o
From	gc	n	e	m	m	m	m	m	m
Hide	R	n	h	o	o	o	o	o	o
Max-Forwards	R	n	e	o	o	o	o	o	o
Organization	g	c	h	-	-	-	o	o	o

Table 4: Summary of header fields, A--O

Other header fields can be added as required; a server MUST ignore header fields not defined in this specification that it does not understand. A proxy MUST NOT remove or modify header fields not defined in this specification that it does not understand. A compact form of these header fields is also defined in Section 9 for use over UDP when the request has to fit into a single packet and size is an issue.

Table 6 in Appendix A lists those header fields that different client and server types MUST be able to parse.

6.1 General Header Fields

General header fields apply to both request and response messages. The "general-header" field names can be extended reliably only in combination with a change in the protocol version. However, new or

	where	enc.	e-e	ACK	BYE	CAN	INV	OPT	REG
Proxy-Authenticate	407	n	h	o	o	o	o	o	o
Proxy-Authorization	R	n	h	o	o	o	o	o	o
Proxy-Require	R	n	h	o	o	o	o	o	o
Priority	R	c	e	-	-	-	o	-	-
Require	R		e	o	o	o	o	o	o
Retry-After	R	c	e	-	-	-	-	-	o
Retry-After	404,480,486	c	e	o	o	o	o	o	o
	503	c	e	o	o	o	o	o	o
	600,603	c	e	o	o	o	o	o	o
Response-Key	R	c	e	-	o	o	o	o	o
Record-Route	R		h	o	o	o	o	o	o
Record-Route	2xx		h	o	o	o	o	o	o
Route	R		h	o	o	o	o	o	o
Server	r	c	e	o	o	o	o	o	o
Subject	R	c	e	-	-	-	o	-	-
Timestamp	g		e	o	o	o	o	o	o
To	gc(1)	n	e	m	m	m	m	m	m
Unsupported	420		e	o	o	o	o	o	o
User-Agent	g	c	e	o	o	o	o	o	o
Via	gc(2)	n	e	m	m	m	m	m	m
Warning	r		e	o	o	o	o	o	o
WWW-Authenticate	401	c	e	o	o	o	o	o	o

Table 5: Summary of header fields, P--Z; (1): copied with possible addition of tag; (2): UAS removes first Via header field

experimental header fields MAY be given the semantics of general header fields if all parties in the communication recognize them to be "general-header" fields. Unrecognized header fields are treated as "entity-header" fields.

6.2 Entity Header Fields

The "entity-header" fields define meta-information about the message-body or, if no body is present, about the resource identified by the request. The term "entity header" is an HTTP 1.1 term where the response body can contain a transformed version of the message body. The original message body is referred to as the "entity". We retain the same terminology for header fields but usually refer to the "message body" rather then the entity as the two are the same in SIP.

6.3 Request Header Fields

The "request-header" fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters of a programming language method invocation.

The "request-header" field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of "request-header" fields if all parties in the communication recognize them to be request-header fields. Unrecognized header fields are treated as "entity-header" fields.

6.4 Response Header Fields

The "response-header" fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Response-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of "response-header" fields if all parties in the communication recognize them to be "response-header" fields. Unrecognized header fields are treated as "entity-header" fields.

6.5 End-to-end and Hop-by-hop Headers

End-to-end headers MUST be transmitted unmodified across all proxies, while hop-by-hop headers MAY be modified or added by proxies.

6.6 Header Field Format

Header fields ("general-header", "request-header", "response-header", and "entity-header") follow the same generic header format as that given in Section 3.1 of RFC 822 [24]. Each header field consists of a name followed by a colon (":") and the field value. Field names are case-insensitive. The field value MAY be preceded by any amount of leading white space (LWS), though a single space (SP) is preferred. Header fields can be extended over multiple lines by preceding each extra line with at least one SP or horizontal tab (HT). Applications MUST follow HTTP "common form" when generating these constructs, since there might exist some implementations that fail to accept anything beyond the common forms.

```
message-header  =  field-name ":" [ field-value ] CRLF
field-name      =  token
field-value     =  *( field-content | LWS )
field-content   =  < the OCTETs  making up the field-value
                     and consisting of either *TEXT-UTF8
                     or combinations of token,
                     separators, and quoted-string>
```

The relative order of header fields with different field names is not significant. Multiple header fields with the same field-name may be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list (i.e., #(values)). It MUST be possible to combine the multiple header fields into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The order in which header fields with the same field-name are received is therefore significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

Field names are not case-sensitive, although their values may be.

6.7 Accept

The Accept header follows the syntax defined in [H14.1]. The semantics are also identical, with the exception that if no Accept header is present, the server SHOULD assume a default value of application/sdp.

This request-header field is used only with the INVITE, OPTIONS and REGISTER request methods to indicate what media types are acceptable in the response.

Example:

```
Accept: application/sdp;level=1, application/x-private, text/html
```

6.8 Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but restricts the content-codings [H3.4.1] that are acceptable in the response. See [H14.3]. The syntax of this header is defined in [H14.3]. The semantics in SIP are identical to those defined in [H14.3].

6.9 Accept-Language

The Accept-Language header follows the syntax defined in [H14.4]. The rules for ordering the languages based on the q parameter apply to SIP as well. When used in SIP, the Accept-Language request-header field can be used to allow the client to indicate to the server in which language it would prefer to receive reason phrases, session descriptions or status responses carried as message bodies. A proxy MAY use this field to help select the destination for the call, for example, a human operator conversant in a language spoken by the caller.

Example:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

6.10 Allow

The Allow entity-header field lists the set of methods supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource. An Allow header field MUST be present in a 405 (Method Not Allowed) response and SHOULD be present in an OPTIONS response.

```
Allow  =  "Allow" ":" 1#Method
```

6.11 Authorization

A user agent that wishes to authenticate itself with a server -- usually, but not necessarily, after receiving a 401 response -- MAY do so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

Section 13.2 overviews the use of the Authorization header, and section 15 describes the syntax and semantics when used with PGP based authentication.

6.12 Call-ID

The Call-ID general-header field uniquely identifies a particular invitation or all registrations of a particular client. Note that a single multimedia conference can give rise to several calls with different Call-IDs, e.g., if a user invites a single individual several times to the same (long-running) conference.

For an INVITE request, a callee user agent server SHOULD NOT alert the user if the user has responded previously to the Call-ID in the INVITE request. If the user is already a member of the conference and the conference parameters contained in the session description have not changed, a callee user agent server MAY silently accept the call, regardless of the Call-ID. An invitation for an existing Call-ID or session can change the parameters of the conference. A client application MAY decide to simply indicate to the user that the conference parameters have been changed and accept the invitation automatically or it MAY require user confirmation.

A user may be invited to the same conference or call using several different Call-IDs. If desired, the client MAY use identifiers within the session description to detect this duplication. For example, SDP contains a session id and version number in the origin (o) field.

The REGISTER and OPTIONS methods use the Call-ID value to unambiguously match requests and responses. All REGISTER requests issued by a single client SHOULD use the same Call-ID, at least within the same boot cycle.

> Since the Call-ID is generated by and for SIP, there is no reason to deal with the complexity of URL-encoding and case-ignoring string comparison.

```
Call-ID   =  ( "Call-ID" | "i" ) ":" local-id "@" host
local-id  =  1*uric
```

"host" SHOULD be either a fully qualified domain name or a globally routable IP address. If this is the case, the "local-id" SHOULD be an identifier consisting of URI characters that is unique within "host". Use of cryptographically random identifiers [27] is RECOMMENDED. If, however, host is not an FQDN or globally routable IP address (such as a net 10 address), the local-id MUST be globally unique, as opposed

to unique within host. These rules guarantee overall global uniqueness of the Call-ID. The value for Call-ID MUST NOT be reused for a different call. Call-IDs are case-sensitive.

Using cryptographically random identifiers provides some protection against session hijacking. Call-ID, To and From are needed to identify a call leg. The distinction between call and call leg matters in calls with third-party control.

For systems which have tight bandwidth constraints, many of the mandatory SIP headers have a compact form, as discussed in Section 9. These are alternate names for the headers which occupy less space in the message. In the case of Call-ID, the compact form is i.

For example, both of the following are valid:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com
```

or

```
i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com
```

6.13 Contact

The Contact general-header field can appear in INVITE, ACK, and REGISTER requests, and in 1xx, 2xx, 3xx, and 485 responses. In general, it provides a URL where the user can be reached for further communications.

INVITE and ACK requests: INVITE and ACK requests MAY contain Contact headers indicating from which location the request is originating.

This allows the callee to send future requests, such as BYE, directly to the caller instead of through a series of proxies. The Via header is not sufficient since the desired address may be that of a proxy.

INVITE 2xx responses: A user agent server sending a definitive, positive response (2xx) MAY insert a Contact response header field indicating the SIP address under which it is reachable most directly for future SIP requests, such as ACK, within the

same Call-ID. The Contact header field contains the address of the server itself or that of a proxy, e.g., if the host is behind a firewall. The value of this Contact header is copied into the Request-URI of subsequent requests for this call if the response did not also contain a Record-Route header. If the response also contains a Record-Route header field, the address in the Contact header field is added as the last item in the Route header field. See Section 6.29 for details.

The Contact value SHOULD NOT be cached across calls, as it may not represent the most desirable location for a particular destination address.

INVITE 1xx responses: A UAS sending a provisional response (1xx) MAY insert a Contact response header. It has the same semantics in a 1xx response as a 2xx INVITE response. Note that CANCEL requests MUST NOT be sent to that address, but rather follow the same path as the original request.

REGISTER requests: REGISTER requests MAY contain a Contact header field indicating at which locations the user is reachable. The REGISTER request defines a wildcard Contact field, "*", which MUST only be used with Expires: 0 to remove all registrations for a particular user. An optional "expires" parameter indicates the desired expiration time of the registration. If a Contact entry does not have an "expires" parameter, the Expires header field is used as the default value. If neither of these mechanisms is used, SIP URIs are assumed to expire after one hour. Other URI schemes have no expiration times.

REGISTER 2xx responses: A REGISTER response MAY return all locations at which the user is currently reachable. An optional "expires" parameter indicates the expiration time of the registration. If a Contact entry does not have an "expires" parameter, the value of the Expires header field indicates the expiration time. If neither mechanism is used, the expiration time specified in the request, explicitly or by default, is used.

3xx and 485 responses: The Contact response-header field can be used with a 3xx or 485 (Ambiguous) response codes to indicate one or more alternate addresses to try. It can appear in responses to BYE, INVITE and OPTIONS methods. The Contact header field contains URIs giving the new locations or user names to try, or may simply specify additional transport parameters. A 300 (Multiple Choices), 301 (Moved Permanently), 302 (Moved Temporarily) or 485 (Ambiguous) response SHOULD contain a Contact field containing URIs of new addresses to be tried. A

301 or 302 response may also give the same location and username that was being tried but specify additional transport parameters such as a different server or multicast address to try or a change of SIP transport from UDP to TCP or vice versa. The client copies the "user", "password", "host", "port" and "user-param" elements of the Contact URI into the Request-URI of the redirected request and directs the request to the address specified by the "maddr" and "port" parameters, using the transport protocol given in the "transport" parameter. If "maddr" is a multicast address, the value of "ttl" is used as the time-to-live value.

Note that the Contact header field MAY also refer to a different entity than the one originally called. For example, a SIP call connected to GSTN gateway may need to deliver a special information announcement such as "The number you have dialed has been changed."

A Contact response header field can contain any suitable URI indicating where the called party can be reached, not limited to SIP URLs. For example, it could contain URL's for phones, fax, or irc (if they were defined) or a mailto: (RFC 2368, [28]) URL.

The following parameters are defined. Additional parameters may be defined in other specifications.

q: The "qvalue" indicates the relative preference among the locations given. "qvalue" values are decimal numbers from 0 to 1, with higher values indicating higher preference.

action: The "action" parameter is used only when registering with the REGISTER request. It indicates whether the client wishes that the server proxy or redirect future requests intended for the client. If this parameter is not specified the action taken depends on server configuration. In its response, the registrar SHOULD indicate the mode used. This parameter is ignored for other requests.

expires: The "expires" parameter indicates how long the URI is valid. The parameter is either a number indicating seconds or a quoted string containing a SIP-date. If this parameter is not provided, the value of the Expires header field determines how long the URI is valid. Implementations MAY treat values larger than 2**32-1 (4294967295 seconds or 136 years) as equivalent to 2**32-1.

```
Contact = ( "Contact" | "m" ) ":"
          ("*" | (1# (( name-addr | addr-spec )
          [ *( ";" contact-params ) ] [ comment ] )))

name-addr      = [ display-name ] "<" addr-spec ">"
addr-spec      = SIP-URL | URI
```

```
display-name   = *token | quoted-string

contact-params = "q"       "=" qvalue
               | "action"  "=" "proxy" | "redirect"
               | "expires" "=" delta-seconds | <"> SIP-date <">
               | extension-attribute

extension-attribute = extension-name [ "=" extension-value ]
```

only allows one address, unquoted. Since URIs can contain commas and semicolons as reserved characters, they can be mistaken for header or parameter delimiters, respectively. The current syntax corresponds to that for the To and From header, which also allows the use of display names.

Example:

```
  Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
     ;q=0.7; expires=3600,
     "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
```

6.14 Content-Encoding

```
Content-Encoding  =  ( "Content-Encoding" | "e" ) ":"
                     1#content-coding
```

The Content-Encoding entity-header field is used as a modifier to the "media-type". When present, its value indicates what additional content codings have been applied to the entity-body, and thus what decoding mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a body to be compressed without losing the identity of its underlying media type.

If multiple encodings have been applied to an entity, the content codings MUST be listed in the order in which they were applied.

All content-coding values are case-insensitive. The Internet Assigned Numbers Authority (IANA) acts as a registry for content-coding value tokens. See [3.5] for a definition of the syntax for content-coding.

Clients MAY apply content encodings to the body in requests. If the server is not capable of decoding the body, or does not recognize any of the content-coding values, it MUST send a 415 "Unsupported Media Type" response, listing acceptable encodings in the Accept-Encoding

header. A server MAY apply content encodings to the bodies in responses. The server MUST only use encodings listed in the Accept-Encoding header in the request.

6.15 Content-Length

The Content-Length entity-header field indicates the size of the message-body, in decimal number of octets, sent to the recipient.

```
Content-Length  =  ( "Content-Length" | "l" ) ":" 1*DIGIT
```

An example is

```
Content-Length: 3495
```

Applications SHOULD use this field to indicate the size of the message-body to be transferred, regardless of the media type of the entity. Any Content-Length greater than or equal to zero is a valid value. If no body is present in a message, then the Content-Length header field MUST be set to zero. If a server receives a UDP request without Content-Length, it MUST assume that the request encompasses the remainder of the packet. If a server receives a UDP request with a Content-Length, but the value is larger than the size of the body sent in the request, the client SHOULD generate a 400 class response. If there is additional data in the UDP packet after the last byte of the body has been read, the server MUST treat the remaining data as a separate message. This allows several messages to be placed in a single UDP packet.

If a response does not contain a Content-Length, the client assumes that it encompasses the remainder of the UDP packet or the data until the TCP connection is closed, as applicable. Section 8 describes how to determine the length of the message body.

6.16 Content-Type

The Content-Type entity-header field indicates the media type of the message-body sent to the recipient. The "media-type" element is defined in [H3.7].

```
Content-Type  =  ( "Content-Type" | "c" ) ":" media-type
```

Examples of this header field are

```
Content-Type: application/sdp
Content-Type: text/html; charset=ISO-8859-4
```

6.17 CSeq

Clients MUST add the CSeq (command sequence) general-header field to every request. A CSeq header field in a request contains the request method and a single decimal sequence number chosen by the requesting client, unique within a single value of Call-ID. The sequence number MUST be expressible as a 32-bit unsigned integer. The initial value of the sequence number is arbitrary, but MUST be less than 2**31. Consecutive requests that differ in request method, headers or body, but have the same Call-ID MUST contain strictly monotonically increasing and contiguous sequence numbers; sequence numbers do not wrap around. Retransmissions of the same request carry the same sequence number, but an INVITE with a different message body or different header fields (a "re-invitation") acquires a new, higher sequence number. A server MUST echo the CSeq value from the request in its response. If the Method value is missing in the received CSeq header field, the server fills it in appropriately.

The ACK and CANCEL requests MUST contain the same CSeq value as the INVITE request that it refers to, while a BYE request cancelling an invitation MUST have a higher sequence number. A BYE request with a CSeq that is not higher should cause a 400 response to be generated.

A user agent server MUST remember the highest sequence number for any INVITE request with the same Call-ID value. The server MUST respond to, and then discard, any INVITE request with a lower sequence number.

All requests spawned in a parallel search have the same CSeq value as the request triggering the parallel search.

```
CSeq  =  "CSeq" ":" 1*DIGIT Method
```

Strictly speaking, CSeq header fields are needed for any SIP request that can be cancelled by a BYE or CANCEL request or where a client can issue several requests for the same Call-ID in close succession. Without a sequence

number, the response to an INVITE could be mistaken for the response to the cancellation (BYE or CANCEL). Also, if the network duplicates packets or if an ACK is delayed until the server has sent an additional response, the client could interpret an old response as the response to a re-invitation issued shortly thereafter. Using CSeq also makes it easy for the server to distinguish different versions of an invitation, without comparing the message body.

The Method value allows the client to distinguish the response to an INVITE request from that of a CANCEL response. CANCEL requests can be generated by proxies; if they were to increase the sequence number, it might conflict with a later request issued by the user agent for the same call.

With a length of 32 bits, a server could generate, within a single call, one request a second for about 136 years before needing to wrap around. The initial value of the sequence number is chosen so that subsequent requests within the same call will not wrap around. A non-zero initial value allows to use a time-based initial sequence number, if the client desires. A client could, for example, choose the 31 most significant bits of a 32-bit second clock as an initial sequence number.

Forked requests MUST have the same CSeq as there would be ambiguity otherwise between these forked requests and later BYE issued by the client user agent.

Example:

```
CSeq: 4711 INVITE
```

6.18 Date

Date is a general-header field. Its syntax is:

```
SIP-date  =  rfc1123-date
```

See [H14.19] for a definition of rfc1123-date. Note that unlike HTTP/1.1, SIP only supports the most recent RFC1123 [29] formatting for dates.

The Date header field reflects the time when the request or response is first sent. Thus, retransmissions have the same Date header field value as the original.

> The Date header field can be used by simple end systems without a battery-backed clock to acquire a notion of current time.

6.19 Encryption

The Encryption general-header field specifies that the content has been encrypted. Section 13 describes the overall SIP security architecture and algorithms. This header field is intended for end-to-end encryption of requests and responses. Requests are encrypted based on the public key belonging to the entity named in the To header field. Responses are encrypted based on the public key conveyed in the Response-Key header field. Note that the public keys themselves may not be used for the encryption. This depends on the particular algorithms used.

For any encrypted message, at least the message body and possibly other message header fields are encrypted. An application receiving a request or response containing an Encryption header field decrypts the body and then concatenates the plaintext to the request line and headers of the original message. Message headers in the decrypted part completely replace those with the same field name in the plaintext part. (Note: If only the body of the message is to be encrypted, the body has to be prefixed with CRLF to allow proper concatenation.) Note that the request method and Request-URI cannot be encrypted.

> Encryption only provides privacy; the recipient has no guarantee that the request or response came from the party listed in the From message header, only that the sender used the recipient's public key. However, proxies will not be able to modify the request or response.

```
Encryption         =  "Encryption" ":" encryption-scheme 1*SP
                      #encryption-params
encryption-scheme  =  token
encryption-params  =  token "=" ( token | quoted-string )
```

> The token indicates the form of encryption used; it is described in section 13.

The example in Figure 10 shows a message encrypted with ASCII-armored PGP that was generated by applying "pgp -ea" to the payload to be encrypted.

```
INVITE sip:watson@boston.bell-telephone.com SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
From: <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Content-Length: 885
Encryption: PGP version=2.6.2,encoding=ascii

hQEMAxkp5GPd+j5xAQf/ZDIfGD/PDOM1wayvwdQAKgGgjmZWe+MTy9NEX8O25Red
h0/pyrd/+DV5C2BYs7yzSOSXaj1C/tTK/4do6rtjhP8QA3vbDdVdaFciwEVAcuXs
ODxlNAVqyDi1RqFC28BJIvQ5KfEkPuACKTK7WlRSBc7vNPEA3nyqZGBTwhxRSbIR
RuFEsHSVojdCam4htcqxGnFwD9sksqs6LIyCFaiTAhWtwcCaN437G7mUYzy2KLcA
zPVGq1VQg83b99zPzIxRdlZ+K7+bAnu8Rtu+ohOCMLV3TPXbyp+err1YiThCZHIu
X9dOVj3CMjCP66RSHa/ea0wYTRRNYA/G+kdP8DSUcqYAAAE/hZPX6nFIqk7AVnf6
IpWHUPTelNUJpzUp5Ou+q/5P7ZAsn+cSAuF2YWtVjCf+SQmBR13p2EYYWHoxlA2/
GgKADYe4M3JSwOtqwU8zUJF3FIfk7vsxmSqtUQrRQaiIhqNyG7KxJt4YjWnEjF5E
WUIPhvyGFMJaeQXIyGRYZAYvKKklyAJcm29zLACxU5alX4M25lHQd9FR9Zmq6Jed
wbWvia6cAIfsvlZ9JGocmQYF7pcuz5pnczqP+/yvRqFJtDGD/v3s++G2R+ViVYJO
z/lxGUZaM4IWBCf+4DUjNanZM0oxAE28NjaIZ0rrldDQmO8V9FtPKdHxkqA5iJP+
6vGOFti1Ak4kmEz0vM/Nsv7kkubTFhRl05OiJIGr9S1UhenlZv9l6RuXsOY/EwH2
z8X9N4MhMyXEVuC9rt8/AUhmVQ==
=bOW+
```

Figure 10: PGP Encryption Example

Since proxies can base their forwarding decision on any combination of SIP header fields, there is no guarantee that an encrypted request "hiding" header fields will reach the same destination as an otherwise identical un-encrypted request.

6.20 Expires

The Expires entity-header field gives the date and time after which the message content expires.

This header field is currently defined only for the REGISTER and INVITE methods. For REGISTER, it is a request and response-header field. In a REGISTER request, the client indicates how long it wishes the registration to be valid. In the response, the server indicates

the earliest expiration time of all registrations. The server MAY choose a shorter time interval than that requested by the client, but SHOULD NOT choose a longer one.

For INVITE requests, it is a request and response-header field. In a request, the caller can limit the validity of an invitation, for example, if a client wants to limit the time duration of a search or a conference invitation. A user interface MAY take this as a hint to leave the invitation window on the screen even if the user is not currently at the workstation. This also limits the duration of a search. If the request expires before the search completes, the proxy returns a 408 (Request Timeout) status. In a 302 (Moved Temporarily) response, a server can advise the client of the maximal duration of the redirection.

The value of this field can be either a SIP-date or an integer number of seconds (in decimal), measured from the receipt of the request. The latter approach is preferable for short durations, as it does not depend on clients and servers sharing a synchronized clock. Implementations MAY treat values larger than 2**32-1 (4294967295 or 136 years) as equivalent to 2**32-1.

```
Expires  =  "Expires" ":" ( SIP-date | delta-seconds )
```

Two examples of its use are

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Expires: 5
```

6.21 From

Requests and responses MUST contain a From general-header field, indicating the initiator of the request. The From field MAY contain the "tag" parameter. The server copies the From header field from the request to the response. The optional "display-name" is meant to be rendered by a human-user interface. A system SHOULD use the display name "Anonymous" if the identity of the client is to remain hidden.

The SIP-URL MUST NOT contain the "transport-param", "maddr-param", "ttl-param", or "headers" elements. A server that receives a SIP-URL with these elements removes them before further processing.

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or semicolon.

```
From         =  ( "From" | "f" ) ":" ( name-addr | addr-spec )
                *( ";" addr-params )
addr-params  =  tag-param
tag-param    =  "tag=" UUID
UUID         =  1*( hex | "-" )
```

Examples:

```
From: "A. G. Bell" <sip:agb@bell-telephone.com>
From: sip:+12125551212@server.phone2net.com
From: Anonymous <sip:c8oqz84zk7z@privacy.org>
```

The "tag" MAY appear in the From field of a request. It MUST be present when it is possible that two instances of a user sharing a SIP address can make call invitations with the same Call-ID.

The "tag" value MUST be globally unique and cryptographically random with at least 32 bits of randomness. A single user maintains the same tag throughout the call identified by the Call-ID.

Call-ID, To and From are needed to identify a call leg. The distinction between call and call leg matters in calls with multiple responses to a forked request. The format is similar to the equivalent RFC 822 [24] header, but with a URI instead of just an email address.

6.22 Hide

A client uses the Hide request header field to indicate that it wants the path comprised of the Via header fields (Section 6.40) to be hidden from subsequent proxies and user agents. It can take two forms: Hide: route and Hide: hop. Hide header fields are typically added by the client user agent, but MAY be added by any proxy along the path.

If a request contains the "Hide: route" header field, all following proxies SHOULD hide their previous hop. If a request contains the "Hide: hop" header field, only the next proxy SHOULD hide the previous hop and then remove the Hide option unless it also wants to remain anonymous.

A server hides the previous hop by encrypting the "host" and "port" parts of the top-most Via header field with an algorithm of its choice. Servers SHOULD add additional "salt" to the "host" and "port" information prior to encryption to prevent malicious downstream proxies from guessing earlier parts of the path based on seeing identical encrypted Via headers. Hidden Via fields are marked with the "hidden" Via option, as described in Section 6.40.

A server that is capable of hiding Via headers MUST attempt to decrypt all Via headers marked as "hidden" to perform loop detection. Servers that are not capable of hiding can ignore hidden Via fields in their loop detection algorithm.

> If hidden headers were not marked, a proxy would have to decrypt all headers to detect loops, just in case one was encrypted, as the Hide: Hop option may have been removed along the way.

A host MUST NOT add such a "Hide: hop" header field unless it can guarantee it will only send a request for this destination to the same next hop. The reason for this is that it is possible that the request will loop back through this same hop from a downstream proxy. The loop will be detected by the next hop if the choice of next hop is fixed, but could loop an arbitrary number of times otherwise.

A client requesting "Hide: route" can only rely on keeping the request path private if it sends the request to a trusted proxy. Hiding the route of a SIP request is of limited value if the request results in data packets being exchanged directly between the calling and called user agent.

The use of Hide header fields is discouraged unless path privacy is truly needed; Hide fields impose extra processing costs and restrictions for proxies and can cause requests to generate 482 (Loop Detected) responses that could otherwise be avoided.

The encryption of Via header fields is described in more detail in Section 13.

The Hide header field has the following syntax:

```
Hide  =  "Hide" ":" ( "route" | "hop" )
```

6.23 Max-Forwards

The Max-Forwards request-header field may be used with any SIP method to limit the number of proxies or gateways that can forward the request to the next downstream server. This can also be useful when the client is attempting to trace a request chain which appears to be failing or looping in mid-chain.

```
Max-Forwards  =  "Max-Forwards" ":" 1*DIGIT
```

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message is allowed to be forwarded.

Each proxy or gateway recipient of a request containing a Max-Forwards header field MUST check and update its value prior to forwarding the request. If the received value is zero (0), the recipient MUST NOT forward the request. Instead, for the OPTIONS and REGISTER methods, it MUST respond as the final recipient. For all other methods, the server returns 483 (Too many hops).

If the received Max-Forwards value is greater than zero, then the forwarded message MUST contain an updated Max-Forwards field with a value decremented by one (1).

Example:

```
Max-Forwards: 6
```

6.24 Organization

The Organization general-header field conveys the name of the organization to which the entity issuing the request or response belongs. It MAY also be inserted by proxies at the boundary of an organization.

The field MAY be used by client software to filter calls.

```
Organization  =  "Organization" ":" *TEXT-UTF8
```

6.25 Priority

The Priority request-header field indicates the urgency of the request as perceived by the client.

```
Priority        =  "Priority" ":" priority-value
priority-value  =  "emergency" | "urgent" | "normal"
                |  "non-urgent"
```

It is RECOMMENDED that the value of "emergency" only be used when life, limb or property are in imminent danger.

Examples:

```
Subject: A tornado is heading our way!
Priority: emergency

Subject: Weekend plans
Priority: non-urgent
```

These are the values of RFC 2076 [30], with the addition of "emergency".

6.26 Proxy-Authenticate

The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI.

Unlike its usage within HTTP, the Proxy-Authenticate header MUST be passed upstream in the response to the UAC. In SIP, only UAC's can authenticate themselves to proxies.

The syntax for this header is defined in [H14.33]. See 14 for further details on its usage.

A client SHOULD cache the credentials used for a particular proxy server and realm for the next request to that server. Credentials are, in general, valid for a specific value of the Request-URI at a particular proxy server. If a client contacts a proxy server that has required authentication in the past, but the client does not have credentials for the particular Request-URI, it MAY attempt to use the most-recently used credential. The server responds with 401 (Unauthorized) if the client guessed wrong.

> This suggested caching behavior is motivated by proxies restricting phone calls to authenticated users. It seems likely that in most cases, all destinations require the same password. Note that end-to-end authentication is likely to be destination-specific.

6.27 Proxy-Authorization

The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy which requires authentication. The Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

Unlike Authorization, the Proxy-Authorization header field applies only to the next outbound proxy that demanded authentication using the Proxy- Authenticate field. When multiple proxies are used in a chain, the Proxy-Authorization header field is consumed by the first outbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

See [H14.34] for a definition of the syntax, and section 14 for a discussion of its usage.

6.28 Proxy-Require

The Proxy-Require header field is used to indicate proxy-sensitive features that MUST be supported by the proxy. Any Proxy-Require header field features that are not supported by the proxy MUST be negatively acknowledged by the proxy to the client if not supported. Proxy servers treat this field identically to the Require field.

See Section 6.30 for more details on the mechanics of this message and a usage example.

6.29 Record-Route

The Record-Route request and response header field is added to a request by any proxy that insists on being in the path of subsequent requests for the same call leg. It contains a globally reachable Request-URI that identifies the proxy server. Each proxy server adds its Request-URI to the beginning of the list.

The server copies the Record-Route header field unchanged into the response. (Record-Route is only relevant for 2xx responses.)

The calling user agent client copies the Record-Route header into a Route header field of subsequent requests within the same call leg, reversing the order of requests, so that the first entry is closest to the user agent client. If the response contained a Contact header field, the calling user agent adds its content as the last Route header. Unless this would cause a loop, any client MUST send any subsequent requests for this call leg to the first Request-URI in the Route request header field and remove that entry.

The calling user agent MUST NOT use the Record-Route header field in requests that contain Route header fields.

> Some proxies, such as those controlling firewalls or in an automatic call distribution (ACD) system, need to maintain call state and thus need to receive any BYE and ACK packets for the call.

The Record-Route header field has the following syntax:

```
Record-Route  =  "Record-Route" ":" 1# name-addr
```

Proxy servers SHOULD use the "maddr" URL parameter containing their address to ensure that subsequent requests are guaranteed to reach exactly the same server.

Example for a request that has traversed the hosts ieee.org and bell-telephone.com , in that order:

```
Record-Route: <sip:a.g.bell@bell-telephone.com>,
  <sip:a.bell@ieee.org>
```

6.30 Require

The Require request-header field is used by clients to tell user agent servers about options that the client expects the server to support in order to properly process the request. If a server does not understand the option, it MUST respond by returning status code 420 (Bad Extension) and list those options it does not understand in the Unsupported header.

```
Require  =  "Require" ":" 1#option-tag
```

Example:

```
C->S:   INVITE sip:watson@bell-telephone.com SIP/2.0
        Require: com.example.billing
        Payment: sheep_skins, conch_shells

S->C:   SIP/2.0 420 Bad Extension
        Unsupported: com.example.billing
```

This is to make sure that the client-server interaction will proceed without delay when all options are understood by both sides, and only slow down if options are not understood (as in the example above). For a well-matched client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms. In addition, it also removes ambiguity when the client requires features that the server does not understand. Some features, such as call handling fields, are only of interest to end systems.

Proxy and redirect servers MUST ignore features that are not understood. If a particular extension requires that intermediate devices support it, the extension MUST be tagged in the Proxy-Require field as well (see Section 6.28).

6.31 Response-Key

The Response-Key request-header field can be used by a client to request the key that the called user agent SHOULD use to encrypt the response with. The syntax is:

```
Response-Key  =  "Response-Key" ":" key-scheme 1*SP #key-param
key-scheme    =  token
key-param     =  token "=" ( token | quoted-string )
```

The "key-scheme" gives the type of encryption to be used for the response. Section 13 describes security schemes.

If the client insists that the server return an encrypted response, it includes a

```
Require: org.ietf.sip.encrypt-response
```

header field in its request. If the server cannot encrypt for whatever reason, it MUST follow normal Require header field procedures and return a 420 (Bad Extension) response. If this Require header field is not present, a server SHOULD still encrypt if it can.

6.32 Retry-After

The Retry-After general-header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 600 (Busy), or 603 (Decline) response to indicate when the called party anticipates being available again. The value of this field can be either an SIP-date or an integer number of seconds (in decimal) after the time of the response.

A REGISTER request MAY include this header field when deleting registrations with "Contact: * ;expires: 0". The Retry-After value then indicates when the user might again be reachable. The registrar MAY then include this information in responses to future calls.

An optional comment can be used to indicate additional information about the time of callback. An optional "duration" parameter indicates how long the called party will be reachable starting at the initial time of availability. If no duration parameter is given, the service is assumed to be available indefinitely.

```
Retry-After  =  "Retry-After" ":" ( SIP-date | delta-seconds )
                [ comment ] [ ";" "duration" "=" delta-seconds ]
```

Examples of its use are

```
  Retry-After: Mon, 21 Jul 1997 18:48:34 GMT (I'm in a meeting)
```

```
Retry-After: Mon, 01 Jan 9999 00:00:00 GMT
  (Dear John: Don't call me back, ever)
Retry-After: Fri, 26 Sep 1997 21:00:00 GMT;duration=3600
Retry-After: 120
```

In the third example, the callee is reachable for one hour starting at 21:00 GMT. In the last example, the delay is 2 minutes.

6.33 Route

The Route request-header field determines the route taken by a request. Each host removes the first entry and then proxies the request to the host listed in that entry, also using it as the Request-URI. The operation is further described in Section 6.29.

The Route header field has the following syntax:

```
Route  =  "Route" ":" 1# name-addr
```

6.34 Server

The Server response-header field contains information about the software used by the user agent server to handle the request. The syntax for this field is defined in [H14.39].

6.35 Subject

This is intended to provide a summary, or to indicate the nature, of the call, allowing call filtering without having to parse the session description. (Also, the session description does not have to use the same subject indication as the invitation.)

```
Subject  =  ( "Subject" | "s" ) ":" *TEXT-UTF8
```

Example:

```
Subject: Tune in - they are talking about your work!
```

6.36 Timestamp

The timestamp general-header field describes when the client sent the request to the server. The value of the timestamp is of significance only to the client and it MAY use any timescale. The server MUST echo the exact same value and MAY, if it has accurate information about this, add a floating point number indicating the number of seconds that have elapsed since it has received the request. The timestamp is used by the client to compute the round-trip time to the server so that it can adjust the timeout value for retransmissions.

```
Timestamp  =  "Timestamp" ":" *(DIGIT) [ "." *(DIGIT) ] [ delay ]
delay      =  *(DIGIT) [ "." *(DIGIT) ]
```

Note that there MUST NOT be any LWS between a DIGIT and the decimal point.

6.37 To

The To general-header field specifies recipient of the request, with the same SIP URL syntax as the From field.

```
To  =  ( "To" | "t" ) ":" ( name-addr | addr-spec )
       *( ";" addr-params )
```

Requests and responses MUST contain a To general-header field, indicating the desired recipient of the request. The optional "display-name" is meant to be rendered by a human-user interface. The UAS or redirect server copies the To header field into its response, and MUST add a "tag" parameter if the request contained more than one Via header field.

> If there was more than one Via header field, the request was handled by at least one proxy server. Since the receiver cannot know whether any of the proxy servers forked the request, it is safest to assume that they might have.

The SIP-URL MUST NOT contain the "transport-param", "maddr-param", "ttl-param", or "headers" elements. A server that receives a SIP-URL with these elements removes them before further processing.

The "tag" parameter serves as a general mechanism to distinguish multiple instances of a user identified by a single SIP URL. As proxies can fork requests, the same request can reach multiple instances of a user (mobile and home phones, for example). As each can respond, there needs to be a means to distinguish the responses from each at the caller. The situation also arises with multicast requests. The tag in the To header field serves to distinguish responses at the UAC. It MUST be placed in the To field of the response by each instance when there is a possibility that the request was forked at an intermediate proxy. The "tag" MUST be added by UAS, registrars and redirect servers, but MUST NOT be inserted into responses forwarded upstream by proxies. The "tag" is added for all definitive responses for all methods, and MAY be added for informational responses from a UAS or redirect server. All subsequent transactions between two entities MUST include the "tag" parameter, as described in Section 11.

See Section 6.21 for details of the "tag" parameter.

The "tag" parameter in To headers is ignored when matching responses to requests that did not contain a "tag" in their To header.

A SIP server returns a 400 (Bad Request) response if it receives a request with a To header field containing a URI with a scheme it does not recognize.

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or semicolon.

The following are examples of valid To headers:

```
To: The Operator <sip:operator@cs.columbia.edu>;tag=287447
To: sip:+12125551212@server.phone2net.com
```

Call-ID, To and From are needed to identify a call leg. The distinction between call and call leg matters in calls with multiple responses from a forked request. The "tag" is added to the To header field in the response to allow forking of future requests for the same call by proxies, while addressing only one of the possibly several responding user agent servers. It also allows several instances of the callee to send requests that can be distinguished.

6.38 Unsupported

The Unsupported response-header field lists the features not supported by the server. See Section 6.30 for a usage example and motivation.

Syntax:

```
Unsupported  =  "Unsupported" ":" 1#option-tag
```

6.39 User-Agent

The User-Agent general-header field contains information about the client user agent originating the request. The syntax and semantics are defined in [H14.42].

6.40 Via

The Via field indicates the path taken by the request so far. This prevents request looping and ensures replies take the same path as the requests, which assists in firewall traversal and other unusual routing situations.

6.40.1 Requests

The client originating the request MUST insert into the request a Via field containing its host name or network address and, if not the default port number, the port number at which it wishes to receive responses. (Note that this port number can differ from the UDP source port number of the request.) A fully-qualified domain name is RECOMMENDED. Each subsequent proxy server that sends the request onwards MUST add its own additional Via field before any existing Via fields. A proxy that receives a redirection (3xx) response and then searches recursively, MUST use the same Via headers as on the original proxied request.

A proxy SHOULD check the top-most Via header field to ensure that it contains the sender's correct network address, as seen from that proxy. If the sender's address is incorrect, the proxy MUST add an additional "received" attribute, as described 6.40.2.

> A host behind a network address translator (NAT) or firewall may not be able to insert a network address into the Via header that can be reached by the next hop beyond

the NAT. Use of the received attribute allows SIP requests to traverse NAT's which only modify the source IP address. NAT's which modify port numbers, called Network Address Port Translator's (NAPT) will not properly pass SIP when transported on UDP, in which case an application layer gateway is required. When run over TCP, SIP stands a better chance of traversing NAT's, since its behavior is similar to HTTP in this case (but of course on different ports).

A proxy sending a request to a multicast address MUST add the "maddr" parameter to its Via header field, and SHOULD add the "ttl" parameter. If a server receives a request which contained an "maddr" parameter in the topmost Via field, it SHOULD send the response to the multicast address listed in the "maddr" parameter.

If a proxy server receives a request which contains its own address in the Via header value, it MUST respond with a 482 (Loop Detected) status code.

A proxy server MUST NOT forward a request to a multicast group which already appears in any of the Via headers.

This prevents a malfunctioning proxy server from causing loops. Also, it cannot be guaranteed that a proxy server can always detect that the address returned by a location service refers to a host listed in the Via list, as a single host may have aliases or several network interfaces.

6.40.2 Receiver-tagged Via Header Fields

Normally, every host that sends or forwards a SIP message adds a Via field indicating the path traversed. However, it is possible that Network Address Translators (NATs) changes the source address and port of the request (e.g., from net-10 to a globally routable address), in which case the Via header field cannot be relied on to route replies. To prevent this, a proxy SHOULD check the top-most Via header field to ensure that it contains the sender's correct network address, as seen from that proxy. If the sender's address is incorrect, the proxy MUST add a "received" parameter to the Via header field inserted by the previous hop. Such a modified Via header field is known as a receiver-tagged Via header field. An example is:

```
Via: SIP/2.0/UDP erlang.bell-telephone.com:5060
Via: SIP/2.0/UDP 10.0.0.1:5060 ;received=199.172.136.3
```

In this example, the message originated from 10.0.0.1 and traversed a NAT with the external address border.ieee.org (199.172.136.3) to reach erlang.bell-telephone.com. The latter noticed the mismatch, and added a parameter to the previous hop's Via header field, containing the address that the packet actually came from. (Note that the NAT border.ieee.org is not a SIP server.)

6.40.3 Responses

Via header fields in responses are processed by a proxy or UAC according to the following rules:

1. The first Via header field should indicate the proxy or client processing this response. If it does not, discard the message. Otherwise, remove this Via field.

2. If there is no second Via header field, this response is destined for this client. Otherwise, the processing depends on whether the Via field contains a "maddr" parameter or is a receiver-tagged field:

   - If the second Via header field contains a "maddr" parameter, send the response to the multicast address listed there, using the port indicated in "sent-by", or port 5060 if none is present. The response SHOULD be sent using the TTL indicated in the "ttl" parameter, or with a TTL of 1 if that parameter is not present. For robustness, responses MUST be sent to the address indicated in the "maddr" parameter even if it is not a multicast address.

   - If the second Via header field does not contain a "maddr" parameter and is a receiver-tagged field (Section 6.40.2), send the message to the address in the "received" parameter, using the port indicated in the "sent-by" value, or using port 5060 if none is present.

   - If neither of the previous cases apply, send the message to the address indicated by the "sent-by" value in the second Via header field.

6.40.4 User Agent and Redirect Servers

A UAS or redirect server sends a response based on one of the following rules:

- If the first Via header field in the request contains a "maddr" parameter, send the response to the multicast address

listed there, using the port indicated in "sent-by", or port 5060 if none is present. The response SHOULD be sent using the TTL indicated in the "ttl" parameter, or with a TTL of 1 if that parameter is not present. For robustness, responses MUST be sent to the address indicated in the "maddr" parameter even if it is not a multicast address.

- If the address in the "sent-by" value of the first Via field differs from the source address of the packet, send the response to the actual packet source address, similar to the treatment for receiver-tagged Via header fields (Section 6.40.2).

- If neither of these conditions is true, send the response to the address contained in the "sent-by" value. If the request was sent using TCP, use the existing TCP connection if available.

6.40.5 Syntax

The format for a Via header field is shown in Fig. 11. The defaults for "protocol-name" and "transport" are "SIP" and "UDP", respectively. The "maddr" parameter, designating the multicast address, and the "ttl" parameter, designating the time-to-live (TTL) value, are included only if the request was sent via multicast. The "received" parameter is added only for receiver-added Via fields (Section 6.40.2). For reasons of privacy, a client or proxy may wish to hide its Via information by encrypting it (see Section 6.22). The "hidden" parameter is included if this header field was hidden by the upstream proxy (see 6.22). Note that privacy of the proxy relies on the cooperation of the next hop, as the next-hop proxy will, by necessity, know the IP address and port number of the source host.

The "branch" parameter is included by every forking proxy. The token MUST be unique for each distinct request generated when a proxy forks. CANCEL requests MUST have the same branch value as the corresponding forked request. When a response arrives at the proxy it can use the branch value to figure out which branch the response corresponds to. A proxy which generates a single request (non-forking) MAY also insert the "branch" parameter. The identifier has to be unique only within a set of isomorphic requests.

```
Via: SIP/2.0/UDP first.example.com:4000;ttl=16
  ;maddr=224.2.0.1 ;branch=a7c6a8dlze (Example)
Via: SIP/2.0/UDP adk8
```

```
Via              = ( "Via" | "v") ":" 1#( sent-protocol sent-by
                   *( ";" via-params ) [ comment ] )
via-params       = via-hidden | via-ttl | via-maddr
                 | via-received | via-branch
via-hidden       = "hidden"
via-ttl          = "ttl" "=" ttl
via-maddr        = "maddr" "=" maddr
via-received     = "received" "=" host
via-branch       = "branch" "=" token
sent-protocol    = protocol-name "/" protocol-version "/" transport
protocol-name    = "SIP" | token
protocol-version = token
transport        = "UDP" | "TCP" | token
sent-by          = ( host [ ":" port ] ) | ( concealed-host )
concealed-host   = token
ttl              = 1*3DIGIT     ; 0 to 255
```

Figure 11: Syntax of Via header field

6.41 Warning

The Warning response-header field is used to carry additional information about the status of a response. Warning headers are sent with responses and have the following format:

```
Warning        =  "Warning" ":" 1#warning-value
warning-value  =  warn-code SP warn-agent SP warn-text
warn-code      =  3DIGIT
warn-agent     =  ( host [ ":" port ] ) | pseudonym
                  ;  the name or pseudonym of the server adding
                  ;  the Warning header, for use in debugging
warn-text      =  quoted-string
```

A response MAY carry more than one Warning header.

The "warn-text" should be in a natural language that is most likely to be intelligible to the human user receiving the response. This decision can be based on any available knowledge, such as the location of the cache or user, the Accept-Language field in a request, or the Content-Language field in a response. The default language is i-default [31].

Any server MAY add Warning headers to a response. Proxy servers MUST place additional Warning headers before any Authorization headers. Within that constraint, Warning headers MUST be added after any existing Warning headers not covered by a signature. A proxy server MUST NOT delete any Warning header field that it received with a response.

When multiple Warning headers are attached to a response, the user agent SHOULD display as many of them as possible, in the order that they appear in the response. If it is not possible to display all of the warnings, the user agent first displays warnings that appear early in the response.

The warn-code consists of three digits. A first digit of "3" indicates warnings specific to SIP.

This is a list of the currently-defined "warn-code"s, each with a recommended warn-text in English, and a description of its meaning. Note that these warnings describe failures induced by the session description.

Warnings 300 through 329 are reserved for indicating problems with keywords in the session description, 330 through 339 are warnings related to basic network services requested in the session description, 370 through 379 are warnings related to quantitative QoS parameters requested in the session description, and 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

300 Incompatible network protocol: One or more network protocols contained in the session description are not available.

301 Incompatible network address formats: One or more network address formats contained in the session description are not available.

302 Incompatible transport protocol: One or more transport protocols described in the session description are not available.

303 Incompatible bandwidth units: One or more bandwidth measurement units contained in the session description were not understood.

304 Media type not available: One or more media types contained in the session description are not available.

305 Incompatible media format: One or more media formats contained in the session description are not available.

306 Attribute not understood: One or more of the media attributes in the session description are not supported.

307 Session description parameter not understood: A parameter other than those listed above was not understood.

330 Multicast not available: The site where the user is located does not support multicast.

331 Unicast not available: The site where the user is located does not support unicast communication (usually due to the presence of a firewall).

370 Insufficient bandwidth: The bandwidth specified in the session description or defined by the media exceeds that known to be available.

399 Miscellaneous warning: The warning text can include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action.

1xx and 2xx have been taken by HTTP/1.1.

Additional "warn-code"s, as in the example below, can be defined through IANA.

Examples:

```
Warning: 307 isi.edu "Session parameter 'foo' not understood"
Warning: 301 isi.edu "Incompatible network address type 'E.164'"
```

6.42 WWW-Authenticate

The WWW-Authenticate response-header field MUST be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI. See [H14.46] for a definition of the syntax, and section 14 for an overview of usage.

The content of the "realm" parameter SHOULD be displayed to the user. A user agent SHOULD cache the authorization credentials for a given value of the destination (To header) and "realm" and attempt to re-use these values on the next request for that destination.

In addition to the "basic" and "digest" authentication schemes defined in the specifications cited above, SIP defines a new scheme, PGP (RFC 2015, [32]), Section 15. Other schemes, such as S/MIME, are for further study.

7 Status Code Definitions

The response codes are consistent with, and extend, HTTP/1.1 response codes. Not all HTTP/1.1 response codes are appropriate, and only those that are appropriate are given here. Other HTTP/1.1 response codes SHOULD NOT be used. Response codes not defined by HTTP/1.1 have codes x80 upwards to avoid clashes with future HTTP response codes. Also, SIP defines a new class, 6xx. The default behavior for unknown response codes is given for each category of codes.

7.1 Informational 1xx

Informational responses indicate that the server or proxy contacted is performing some further action and does not yet have a definitive response. The client SHOULD wait for a further response from the server, and the server SHOULD send such a response without further prompting. A server SHOULD send a 1xx response if it expects to take more than 200 ms to obtain a final response. A server MAY issue zero or more 1xx responses, with no restriction on their ordering or uniqueness. Note that 1xx responses are not transmitted reliably, that is, they do not cause the client to send an ACK. Servers are free to retransmit informational responses and clients can inquire about the current state of call processing by re-sending the request.

7.1.1 100 Trying

Some unspecified action is being taken on behalf of this call (e.g., a database is being consulted), but the user has not yet been located.

7.1.2 180 Ringing

The called user agent has located a possible location where the user has registered recently and is trying to alert the user.

7.1.3 181 Call Is Being Forwarded

A proxy server MAY use this status code to indicate that the call is being forwarded to a different set of destinations.

7.1.4 182 Queued

The called party is temporarily unavailable, but the callee has decided to queue the call rather than reject it. When the callee becomes available, it will return the appropriate final status response. The reason phrase MAY give further details about the status of the call, e.g., "5 calls queued; expected waiting time is 15 minutes". The server MAY issue several 182 responses to update the caller about the status of the queued call.

7.2 Successful 2xx

The request was successful and MUST terminate a search.

7.2.1 200 OK

The request has succeeded. The information returned with the response depends on the method used in the request, for example:

BYE: The call has been terminated. The message body is empty.

CANCEL: The search has been cancelled. The message body is empty.

INVITE: The callee has agreed to participate; the message body indicates the callee's capabilities.

OPTIONS: The callee has agreed to share its capabilities, included in the message body.

REGISTER: The registration has succeeded. The client treats the message body according to its Content-Type.

7.3 Redirection 3xx

3xx responses give information about the user's new location, or about alternative services that might be able to satisfy the call. They SHOULD terminate an existing search, and MAY cause the initiator to begin a new search if appropriate.

Any redirection (3xx) response MUST NOT suggest any of the addresses in the Via (Section 6.40) path of the request in the Contact header field. (Addresses match if their host and port number match.)

To avoid forwarding loops, a user agent client or proxy MUST check whether the address returned by a redirect server equals an address tried earlier.

7.3.1 300 Multiple Choices

The address in the request resolved to several choices, each with its own specific location, and the user (or user agent) can select a preferred communication end point and redirect its request to that location.

The response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate, if allowed by the Accept request header. The entity format is specified by the media type given in the Content-Type header field. The choices SHOULD also be listed as Contact fields (Section 6.13). Unlike HTTP, the SIP response MAY contain several Contact fields or a list of addresses in a Contact field. User agents MAY use the Contact header field value for automatic redirection or MAY ask the user to confirm a choice. However, this specification does not define any standard for such automatic selection.

This status response is appropriate if the callee can be reached at several different locations and the server cannot or prefers not to proxy the request.

7.3.2 301 Moved Permanently

The user can no longer be found at the address in the Request-URI and the requesting client SHOULD retry at the new address given by the Contact header field (Section 6.13). The caller SHOULD update any local directories, address books and user location caches with this new value and redirect future requests to the address(es) listed.

7.3.3 302 Moved Temporarily

The requesting client SHOULD retry the request at the new address(es) given by the Contact header field (Section 6.13). The duration of the redirection can be indicated through an Expires (Section 6.20) header. If there is no explicit expiration time, the address is only valid for this call and MUST NOT be cached for future calls.

7.3.4 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Contact field. The Contact field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses MUST only be generated by user agent servers.

7.3.5 380 Alternative Service

The call was not successful, but alternative services are possible. The alternative services are described in the message body of the response. Formats for such bodies are not defined here, and may be the subject of future standardization.

7.4 Request Failure 4xx

4xx responses are definite failure responses from a particular server. The client SHOULD NOT retry the same request without modification (e.g., adding appropriate authorization). However, the same request to a different server might be successful.

7.4.1 400 Bad Request

The request could not be understood due to malformed syntax.

7.4.2 401 Unauthorized

The request requires user authentication.

7.4.3 402 Payment Required

Reserved for future use.

7.4.4 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help, and the request SHOULD NOT be repeated.

7.4.5 404 Not Found

The server has definitive information that the user does not exist at the domain specified in the Request-URI. This status is also returned if the domain in the Request-URI does not match any of the domains handled by the recipient of the request.

7.4.6 405 Method Not Allowed

The method specified in the Request-Line is not allowed for the address identified by the Request-URI. The response MUST include an Allow header field containing a list of valid methods for the indicated address.

7.4.7 406 Not Acceptable

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

7.4.8 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client MUST first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (section 6.26) containing a challenge applicable to the proxy for the requested resource. The client MAY repeat the request with a suitable Proxy-Authorization header field (section 6.27). SIP access authentication is explained in section 13.2 and 14.

This status code is used for applications where access to the communication channel (e.g., a telephony gateway) rather than the callee requires authentication.

7.4.9 408 Request Timeout

The server could not produce a response, e.g., a user location, within the time indicated in the Expires request-header field. The client MAY repeat the request without modifications at any later time.

7.4.10 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This response is returned if the action parameter in a REGISTER request conflicts with existing registrations.

7.4.11 410 Gone

The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead.

7.4.12 411 Length Required

The server refuses to accept the request without a defined Content-Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

7.4.13 413 Request Entity Too Large

The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

7.4.14 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.

7.4.15 415 Unsupported Media Type

The server is refusing to service the request because the message body of the request is in a format not supported by the requested resource for the requested method. The server SHOULD return a list of acceptable formats using the Accept, Accept-Encoding and Accept-Language header fields.

7.4.16 420 Bad Extension

The server did not understand the protocol extension specified in a Require (Section 6.30) header field.

7.4.17 480 Temporarily Unavailable

The callee's end system was contacted successfully but the callee is currently unavailable (e.g., not logged in or logged in in such a manner as to preclude communication with the callee). The response MAY indicate a better time to call in the Retry-After header. The user could also be available elsewhere (unbeknownst to this host), thus, this response does not terminate any searches. The reason phrase SHOULD indicate a more precise cause as to why the callee is unavailable. This value SHOULD be setable by the user agent. Status 486 (Busy Here) MAY be used to more precisely indicate a particular reason for the call failure.

This status is also returned by a redirect server that recognizes the user identified by the Request-URI, but does not currently have a valid forwarding location for that user.

7.4.18 481 Call Leg/Transaction Does Not Exist

This status is returned under two conditions: The server received a BYE request that does not match any existing call leg or the server received a CANCEL request that does not match any existing transaction. (A server simply discards an ACK referring to an unknown transaction.)

7.4.19 482 Loop Detected

The server received a request with a Via (Section 6.40) path containing itself.

7.4.20 483 Too Many Hops

The server received a request that contains more Via entries (hops) (Section 6.40) than allowed by the Max-Forwards (Section 6.23) header field.

7.4.21 484 Address Incomplete

The server received a request with a To (Section 6.37) address or Request-URI that was incomplete. Additional information SHOULD be provided.

> This status code allows overlapped dialing. With overlapped dialing, the client does not know the length of the dialing string. It sends strings of increasing lengths, prompting the user for more input, until it no longer receives a 484 status response.

7.4.22 485 Ambiguous

The callee address provided in the request was ambiguous. The response MAY contain a listing of possible unambiguous addresses in Contact headers.

Revealing alternatives can infringe on privacy concerns of the user or the organization. It MUST be possible to configure a server to respond with status 404 (Not Found) or to suppress the listing of possible choices if the request address was ambiguous.

Example response to a request with the URL lee@example.com :

```
485 Ambiguous SIP/2.0
Contact: Carol Lee <sip:carol.lee@example.com>
```

```
Contact: Ping Lee <sip:p.lee@example.com>
Contact: Lee M. Foote <sip:lee.foote@example.com>
```

Some email and voice mail systems provide this functionality. A status code separate from 3xx is used since the semantics are different: for 300, it is assumed that the same person or service will be reached by the choices provided. While an automated choice or sequential search makes sense for a 3xx response, user intervention is required for a 485 response.

7.4.23 486 Busy Here

The callee's end system was contacted successfully but the callee is currently not willing or able to take additional calls. The response MAY indicate a better time to call in the Retry-After header. The user could also be available elsewhere, such as through a voice mail service, thus, this response does not terminate any searches. Status 600 (Busy Everywhere) SHOULD be used if the client knows that no other end system will be able to accept this call.

7.5 Server Failure 5xx

5xx responses are failure responses given when a server itself has erred. They are not definitive failures, and MUST NOT terminate a search if other possible locations remain untried.

7.5.1 500 Server Internal Error

The server encountered an unexpected condition that prevented it from fulfilling the request. The client MAY display the specific error condition, and MAY retry the request after several seconds.

7.5.2 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any user.

7.5.3 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the downstream server it accessed in attempting to fulfill the request.

7.5.4 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header. If no Retry-After is given, the client MUST handle the response as it would for a 500 response.

Note: The existence of the 503 status code does not imply that a server has to use it when becoming overloaded. Some servers MAY wish to simply refuse the connection.

7.5.5 504 Gateway Time-out

The server, while acting as a gateway, did not receive a timely response from the server (e.g., a location server) it accessed in attempting to complete the request.

7.5.6 505 Version Not Supported

The server does not support, or refuses to support, the SIP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, other than with this error message. The response MAY contain an entity describing why that version is not supported and what other protocols are supported by that server. The format for such an entity is not defined here and may be the subject of future standardization.

7.6 Global Failures 6xx

6xx responses indicate that a server has definitive information about a particular user, not just the particular instance indicated in the Request-URI. All further searches for this user are doomed to failure and pending searches SHOULD be terminated.

7.6.1 600 Busy Everywhere

The callee's end system was contacted successfully but the callee is busy and does not wish to take the call at this time. The response MAY indicate a better time to call in the Retry-After header. If the callee does not wish to reveal the reason for declining the call, the callee uses status code 603 (Decline) instead. This status response is returned only if the client knows that no other end point (such as a voice mail system) will answer the request. Otherwise, 486 (Busy Here) should be returned.

7.6.2 603 Decline

The callee's machine was successfully contacted but the user explicitly does not wish to or cannot participate. The response MAY indicate a better time to call in the Retry-After header.

7.6.3 604 Does Not Exist Anywhere

The server has authoritative information that the user indicated in the To request field does not exist anywhere. Searching for the user elsewhere will not yield any results.

7.6.4 606 Not Acceptable

The user's agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable.

A 606 (Not Acceptable) response means that the user wishes to communicate, but cannot adequately support the session described. The 606 (Not Acceptable) response MAY contain a list of reasons in a Warning header field describing why the session described cannot be supported. Reasons are listed in Section 6.41. It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join an already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide whether or not to act on a 606 (Not Acceptable) response.

8 SIP Message Body

8.1 Body Inclusion

Requests MAY contain message bodies unless otherwise noted. Within this specification, the BYE request MUST NOT contain a message body. For ACK, INVITE and OPTIONS, the message body is always a session description. The use of message bodies for REGISTER requests is for further study.

For response messages, the request method and the response status code determine the type and interpretation of any message body. All responses MAY include a body. Message bodies for 1xx responses contain advisory information about the progress of the request. 2xx responses to INVITE requests contain session descriptions. In 3xx responses, the message body MAY contain the description of alternative destinations or services, as described in Section 7.3. For responses with status 400 or greater, the message body MAY

contain additional, human-readable information about the reasons for failure. It is RECOMMENDED that information in 1xx and 300 and greater responses be of type text/plain or text/html

8.2 Message Body Type

The Internet media type of the message body MUST be given by the Content-Type header field. If the body has undergone any encoding (such as compression) then this MUST be indicated by the Content-Encoding header field, otherwise Content-Encoding MUST be omitted. If applicable, the character set of the message body is indicated as part of the Content-Type header-field value.

8.3 Message Body Length

The body length in bytes SHOULD be given by the Content-Length header field. Section 6.15 describes the behavior in detail.

The "chunked" transfer encoding of HTTP/1.1 MUST NOT be used for SIP. (Note: The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator.)

9 Compact Form

When SIP is carried over UDP with authentication and a complex session description, it may be possible that the size of a request or response is larger than the MTU. To address this problem, a more compact form of SIP is also defined by using abbreviations for the common header fields listed below:

short field name	long field name	note
c	Content-Type	
e	Content-Encoding	
f	From	
i	Call-ID	
m	Contact	from "moved"
l	Content-Length	
s	Subject	
t	To	
v	Via	

Thus, the message in section 16.2 could also be written:

```
INVITE sip:schooler@vlsi.caltech.edu SIP/2.0
v:SIP/2.0/UDP 131.215.131.131;maddr=239.128.16.254;ttl=16
v:SIP/2.0/UDP 128.16.64.19
f:sip:mjh@isi.edu
t:sip:schooler@cs.caltech.edu
i:62729-27@128.16.64.19
c:application/sdp
CSeq: 4711 INVITE
l:187

v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 224.2.0.1/127
t=0 0
m=audio 3456 RTP/AVP 0
```

Clients MAY mix short field names and long field names within the same request. Servers MUST accept both short and long field names for requests. Proxies MAY change header fields between their long and short forms, but this MUST NOT be done to fields following an Authorization header.

10 Behavior of SIP Clients and Servers

10.1 General Remarks

SIP is defined so it can use either UDP (unicast or multicast) or TCP as a transport protocol; it provides its own reliability mechanism.

10.1.1 Requests

Servers discard isomorphic requests, but first retransmit the appropriate response. (SIP requests are said to be idempotent , i.e., receiving more than one copy of a request does not change the server state.)

After receiving a CANCEL request from an upstream client, a stateful proxy server MAY send a CANCEL on all branches where it has not yet received a final response.

When a user agent receives a request, it checks the Call-ID against those of in-progress calls. If the Call-ID was found, it compares the tag value of To with the user's tag and rejects the request if the

two do not match. If the From header, including any tag value, matches the value for an existing call leg, the server compares the CSeq header field value. If less than or equal to the current sequence number, the request is a retransmission. Otherwise, it is a new request. If the From header does not match an existing call leg, a new call leg is created.

If the Call-ID was not found, a new call leg is created, with entries for the To, From and Call-ID headers. In this case, the To header field should not have contained a tag. The server returns a response containing the same To value, but with a unique tag added. The tag MAY be omitted if the request contained only one Via header field.

10.1.2 Responses

A server MAY issue one or more provisional responses at any time before sending a final response. If a stateful proxy, user agent server, redirect server or registrar cannot respond to a request with a final response within 200 ms, it SHOULD issue a provisional (1xx) response as soon as possible. Stateless proxies MUST NOT issue provisional responses on their own.

Responses are mapped to requests by the matching To, From, Call-ID, CSeq headers and the branch parameter of the first Via header. Responses terminate request retransmissions even if they have Via headers that cause them to be delivered to an upstream client.

A stateful proxy may receive a response that it does not have state for, that is, where it has no a record of an associated request. If the Via header field indicates that the upstream server used TCP, the proxy actively opens a TCP connection to that address. Thus, proxies have to be prepared to receive responses on the incoming side of passive TCP connections, even though most responses will arrive on the incoming side of an active connection. (An active connection is a TCP connection initiated by the proxy, a passive connection is one accepted by the proxy, but initiated by another entity.)

100 responses SHOULD NOT be forwarded, other 1xx responses MAY be forwarded, possibly after the server eliminates responses with status codes that had already been sent earlier. 2xx responses are forwarded according to the Via header. Once a stateful proxy has received a 2xx response, it MUST NOT forward non-2xx final responses. Responses with status 300 and higher are retransmitted by each stateful proxy until the next upstream proxy sends an ACK (see below for timing details) or CANCEL.

A stateful proxy SHOULD maintain state for at least 32 seconds after the receipt of the first definitive non-200 response, in order to handle retransmissions of the response.

> The 32 second window is given by the maximum retransmission duration of 200-class responses using the default timers, in case the ACK is lost somewhere on the way to the called user agent or the next stateful proxy.

10.2 Source Addresses, Destination Addresses and Connections

10.2.1 Unicast UDP

Responses are returned to the address listed in the Via header field (Section 6.40), not the source address of the request.

> Recall that responses are not generated by the next-hop stateless server, but generated by either a proxy server or the user agent server. Thus, the stateless proxy can only use the Via header field to forward the response.

10.2.2 Multicast UDP

Requests MAY be multicast; multicast requests likely feature a host-independent Request-URI. This request SHOULD be scoped to ensure it is not forwarded beyond the boundaries of the administrative system. This MAY be done with either TTL or administrative scopes[25], depending on what is implemented in the network.

A client receiving a multicast query does not have to check whether the host part of the Request-URI matches its own host or domain name. If the request was received via multicast, the response is also returned via multicast. Responses to multicast requests are multicast with the same TTL as the request, where the TTL is derived from the ttl parameter in the Via header (Section 6.40).

To avoid response implosion, servers MUST NOT answer multicast requests with a status code other than 2xx or 6xx. The server delays its response by a random interval uniformly distributed between zero and one second. Servers MAY suppress responses if they hear a lower-numbered or 6xx response from another group member prior to sending. Servers do not respond to CANCEL requests received via multicast to avoid request implosion. A proxy or UAC SHOULD send a CANCEL on receiving the first 2xx or 6xx response to a multicast request.

Server response suppression is a MAY since it requires a server to violate some basic message processing rules. Lets say A sends a multicast request, and it is received by B,C, and D. B sends a 200 response. The topmost Via field in the response will contain the address of A. C will also receive this response, and could use it to suppress its own response. However, C would normally not examine this response, as the topmost Via is not its own. Normally, a response received with an incorrect topmost Via MUST be dropped, but not in this case. To distinguish this packet from a misrouted or multicast looped packet is fairly complex, and for this reason the procedure is a MAY. The CANCEL, instead, provides a simpler and more standard way to perform response suppression. It is for this reason that the use of CANCEL here is a SHOULD

10.3 TCP

A single TCP connection can serve one or more SIP transactions. A transaction contains zero or more provisional responses followed by one or more final responses. (Typically, transactions contain exactly one final response, but there are exceptional circumstances, where, for example, multiple 200 responses can be generated.)

The client SHOULD keep the connection open at least until the first final response arrives. If the client closes or resets the TCP connection prior to receiving the first final response, the server treats this action as equivalent to a CANCEL request.

This behavior makes it less likely that malfunctioning clients cause a proxy server to keep connection state indefinitely.

The server SHOULD NOT close the TCP connection until it has sent its final response, at which point it MAY close the TCP connection if it wishes to. However, normally it is the client's responsibility to close the connection.

If the server leaves the connection open, and if the client so desires it MAY re-use the connection for further SIP requests or for requests from the same family of protocols (such as HTTP or stream control commands).

If a server needs to return a response to a client and no longer has a connection open to that client, it MAY open a connection to the address listed in the Via header. Thus, a proxy or user agent MUST be prepared to receive both requests and responses on a "passive" connection.

10.4 Reliability for BYE, CANCEL, OPTIONS, REGISTER Requests

10.4.1 UDP

A SIP client using UDP SHOULD retransmit a BYE, CANCEL, OPTIONS, or REGISTER request with an exponential backoff, starting at a T1 second interval, doubling the interval for each packet, and capping off at a T2 second interval. This means that after the first packet is sent, the second is sent T1 seconds later, the next 2*T1 seconds after that, the next 4*T1 seconds after that, and so on, until the interval hits T2. Subsequent retransmissions are spaced by T2 seconds. If the client receives a provisional response, it continues to retransmit the request, but with an interval of T2 seconds. Retransmissions cease when the client has sent a total of eleven packets, or receives a definitive response. Default values for T1 and T2 are 500 ms and 4 s, respectively. Clients MAY use larger values, but SHOULD NOT use smaller ones. Servers retransmit the response upon receipt of a request retransmission. After the server sends a final response, it cannot be sure the client has received the response, and thus SHOULD cache the results for at least 10*T2 seconds to avoid having to, for example, contact the user or location server again upon receiving a request retransmission.

> Use of the exponential backoff is for congestion control purposes. However, the back-off must cap off, since request retransmissions are used to trigger response retransmissions at the server. Without a cap, the loss of a single response could significantly increase transaction latencies.

The value of the initial retransmission timer is smaller than that that for TCP since it is expected that network paths suitable for interactive communications have round-trip times smaller than 500 ms. For congestion control purposes, the retransmission count has to be bounded. Given that most transactions are expected to consist of one request and a few responses, round-trip time estimation is not likely to be very useful. If RTT estimation is desired to more quickly discover a missing final response, each request retransmission needs to be labeled with its own Timestamp (Section 6.36), returned in the response. The server caches the result until it can be sure that the client will not retransmit the same request again.

Each server in a proxy chain generates its own final response to a CANCEL request. The server responds immediately upon receipt of the CANCEL request rather than waiting until it has received final responses from the CANCEL requests it generates.

BYE and OPTIONS final responses are generated by redirect and user agent servers; REGISTER final responses are generated by registrars. Note that in contrast to the reliability mechanism described in Section 10.5, responses to these requests are not retransmitted periodically and not acknowledged via ACK.

10.4.2 TCP

Clients using TCP do not need to retransmit requests.

10.5 Reliability for INVITE Requests

Special considerations apply for the INVITE method.

1. After receiving an invitation, considerable time can elapse before the server can determine the outcome. For example, if the called party is "rung" or extensive searches are performed, delays between the request and a definitive response can reach several tens of seconds. If either caller or callee are automated servers not directly controlled by a human being, a call attempt could be unbounded in time.

2. If a telephony user interface is modeled or if we need to interface to the PSTN, the caller's user interface will provide "ringback", a signal that the callee is being alerted. (The status response 180 (Ringing) MAY be used to initiate ringback.) Once the callee picks up, the caller needs to know so that it can enable the voice path and stop ringback. The callee's response to the invitation could get lost. Unless the response is transmitted reliably, the caller will continue to hear ringback while the callee assumes that the call exists.

3. The client has to be able to terminate an on-going request, e.g., because it is no longer willing to wait for the connection or search to succeed. The server will have to wait several retransmission intervals to interpret the lack of request retransmissions as the end of a call. If the call succeeds shortly after the caller has given up, the callee will "pick up the phone" and not be "connected".

10.5.1 UDP

For UDP, A SIP client SHOULD retransmit a SIP INVITE request with an interval that starts at T1 seconds, and doubles after each packet transmission. The client ceases retransmissions if it receives a provisional or definitive response, or once it has sent a total of 7 request packets.

A server which transmits a provisional response should retransmit it upon reception of a duplicate request. A server which transmits a final response should retransmit it with an interval that starts at T1 seconds, and doubles for each subsequent packet. Response retransmissions cease when any one of the following occurs:

1. An ACK request for the same transaction is received;

2. a BYE request for the same call leg is received;

3. a CANCEL request for the same call leg is received and the final response status was equal or greater to 300;

4. the response has been transmitted 7 times.

Only the user agent client generates an ACK for 2xx final responses, If the response contained a Contact header field, the ACK MAY be sent to the address listed in that Contact header field. If the response did not contain a Contact header, the client uses the same To header field and Request-URI as for the INVITE request and sends the ACK to the same destination as the original INVITE request. ACKs for final responses other than 2xx are sent to the same server that the original request was sent to, using the same Request-URI as the original request. Note, however, that the To header field in the ACK is copied from the response being acknowledged, not the request, and thus MAY additionally contain the tag parameter. Also note than unlike 2xx final responses, a proxy generates an ACK for non-2xx final responses.

The ACK request MUST NOT be acknowledged to prevent a response-ACK feedback loop. Fig. 12 and 13 show the client and server state diagram for invitations.

The mechanism in Sec. 10.4 would not work well for INVITE because of the long delays between INVITE and a final response. If the 200 response were to get lost, the callee would believe the call to exist, but the voice path would

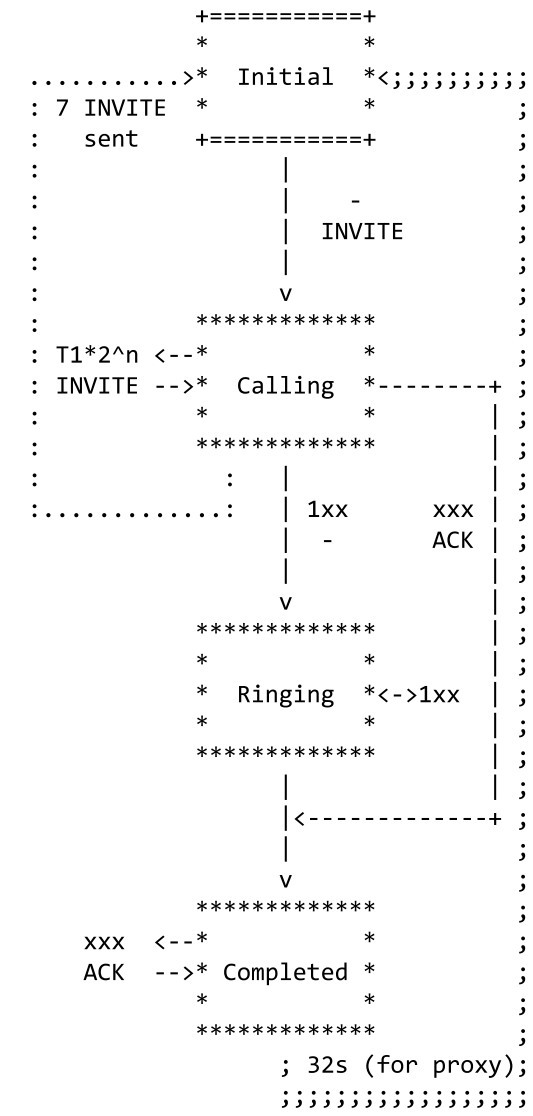


Figure 12: State transition diagram of client for INVITE method

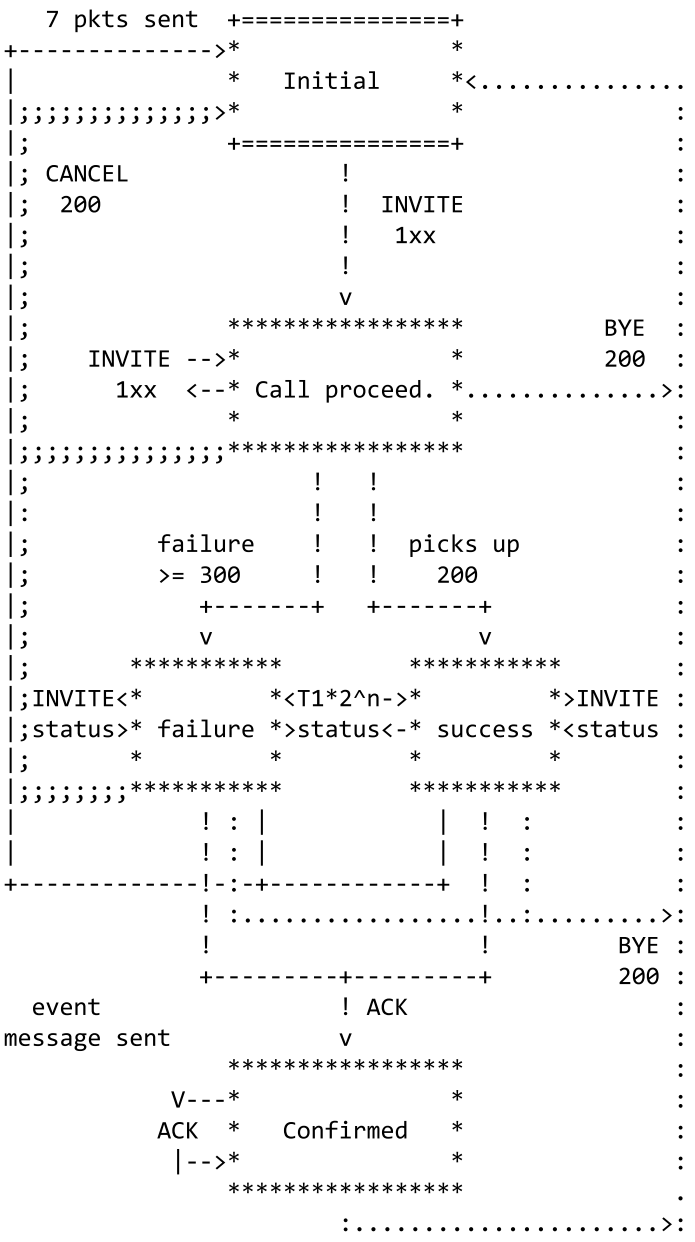


Figure 13: State transition diagram of server for INVITE method

be dead since the caller does not know that the callee has picked up. Thus, the INVITE retransmission interval would have to be on the order of a second or two to limit the duration of this state confusion. Retransmitting the response with an exponential back-off helps ensure that the response is received, without placing an undue burden on the network.

10.5.2 TCP

A user agent using TCP MUST NOT retransmit requests, but uses the same algorithm as for UDP (Section 10.5.1) to retransmit responses until it receives an ACK.

It is necessary to retransmit 2xx responses as their reliability is assured end-to-end only. If the chain of proxies has a UDP link in the middle, it could lose the response, with no possibility of recovery. For simplicity, we also retransmit non-2xx responses, although that is not strictly necessary.

10.6 Reliability for ACK Requests

The ACK request does not generate responses. It is only generated when a response to an INVITE request arrives (see Section 10.5). This behavior is independent of the transport protocol. Note that the ACK request MAY take a different path than the original INVITE request, and MAY even cause a new TCP connection to be opened in order to send it.

10.7 ICMP Handling

Handling of ICMP messages in the case of UDP messages is straightforward. For requests, a host, network, port, or protocol unreachable error SHOULD be treated as if a 400-class response was received. For responses, these errors SHOULD cause the server to cease retransmitting the response.

Source quench ICMP messages SHOULD be ignored. TTL exceeded errors SHOULD be ignored. Parameter problem errors SHOULD be treated as if a 400-class response was received.

11 Behavior of SIP User Agents

This section describes the rules for user agent client and servers for generating and processing requests and responses.

11.1 Caller Issues Initial INVITE Request

When a user agent client desires to initiate a call, it formulates an INVITE request. The To field in the request contains the address of the callee. The Request-URI contains the same address. The From field contains the address of the caller. If the From address can appear in requests generated by other user agent clients for the same call, the caller MUST insert the tag parameter in the From field. A UAC MAY optionally add a Contact header containing an address where it would like to be contacted for transactions from the callee back to the caller.

11.2 Callee Issues Response

When the initial INVITE request is received at the callee, the callee can accept, redirect, or reject the call. In all of these cases, it formulates a response. The response MUST copy the To, From, Call-ID, CSeq and Via fields from the request. Additionally, the responding UAS MUST add the tag parameter to the To field in the response if the request contained more than one Via header field. Since a request from a UAC may fork and arrive at multiple hosts, the tag parameter serves to distinguish, at the UAC, multiple responses from different UAS's. The UAS MAY add a Contact header field in the response. It contains an address where the callee would like to be contacted for subsequent transactions, including the ACK for the current INVITE. The UAS stores the values of the To and From field, including any tags. These become the local and remote addresses of the call leg, respectively.

11.3 Caller Receives Response to Initial Request

Multiple responses may arrive at the UAC for a single INVITE request, due to a forking proxy. Each response is distinguished by the "tag" parameter in the To header field, and each represents a distinct call leg. The caller MAY choose to acknowledge or terminate the call with each responding UAS. To acknowledge, it sends an ACK request, and to terminate it sends a BYE request. The To header field in the ACK or BYE MUST be the same as the To field in the 200 response, including any tag. The From header field MUST be the same as the From header field in the 200 (OK) response, including any tag. The Request-URI of the ACK or BYE request MAY be set to whatever address was found in the Contact header field in the 200 (OK) response, if present. Alternately, a UAC may copy the address from the To header field into the Request-URI. The UAC also notes the value of the To and From header fields in each response. For each call leg, the To header field becomes the remote address, and the From header field becomes the local address.

11.4 Caller or Callee Generate Subsequent Requests

Once the call has been established, either the caller or callee MAY generate INVITE or BYE requests to change or terminate the call. Regardless of whether the caller or callee is generating the new request, the header fields in the request are set as follows. For the desired call leg, the To header field is set to the remote address, and the From header field is set to the local address (both including any tags). The Contact header field MAY be different than the Contact header field sent in a previous response or request. The Request-URI MAY be set to the value of the Contact header field received in a previous request or response from the remote party, or to the value of the remote address.

11.5 Receiving Subsequent Requests

When a request is received subsequently, the following checks are made:

1. If the Call-ID is new, the request is for a new call, regardless of the values of the To and From header fields.

2. If the Call-ID exists, the request is for an existing call. If the To, From, Call-ID, and CSeq values exactly match (including tags) those of any requests received previously, the request is a retransmission.

3. If there was no match to the previous step, the To and From fields are compared against existing call leg local and remote addresses. If there is a match, and the CSeq in the request is higher than the last CSeq received on that leg, the request is a new transaction for an existing call leg.

12 Behavior of SIP Proxy and Redirect Servers

This section describes behavior of SIP redirect and proxy servers in detail. Proxy servers can "fork" connections, i.e., a single incoming request spawns several outgoing (client) requests.

12.1 Redirect Server

A redirect server does not issue any SIP requests of its own. After receiving a request other than CANCEL, the server gathers the list of alternative locations and returns a final response of class 3xx or it refuses the request. For well-formed CANCEL requests, it SHOULD return a 2xx response. This response ends the SIP transaction. The

redirect server maintains transaction state for the whole SIP transaction. It is up to the client to detect forwarding loops between redirect servers.

12.2 User Agent Server

User agent servers behave similarly to redirect servers, except that they also accept requests and can return a response of class 2xx.

12.3 Proxy Server

This section outlines processing rules for proxy servers. A proxy server can either be stateful or stateless. When stateful, a proxy remembers the incoming request which generated outgoing requests, and the outgoing requests. A stateless proxy forgets all information once an outgoing request is generated. A forking proxy SHOULD be stateful. Proxies that accept TCP connections MUST be stateful.

Otherwise, if the proxy were to lose a request, the TCP client would never retransmit it.

A stateful proxy SHOULD NOT become stateless until after it sends a definitive response upstream, and at least 32 seconds after it received a definitive response.

A stateful proxy acts as a virtual UAS/UAC. It implements the server state machine when receiving requests, and the client state machine for generating outgoing requests, with the exception of receiving a 2xx response to an INVITE. Instead of generating an ACK, the 2xx response is always forwarded upstream towards the caller. Furthermore, ACK's for 200 responses to INVITE's are always proxied downstream towards the UAS, as they would be for a stateless proxy.

A stateless proxy does not act as a virtual UAS/UAC (as this would require state). Rather, a stateless proxy forwards every request it receives downstream, and every response it receives upstream.

12.3.1 Proxying Requests

To prevent loops, a server MUST check if its own address is already contained in the Via header field of the incoming request.

The To, From, Call-ID, and Contact tags are copied exactly from the original request. The proxy SHOULD change the Request-URI to indicate the server where it intends to send the request.

A proxy server always inserts a Via header field containing its own address into those requests that are caused by an incoming request. Each proxy MUST insert a "branch" parameter (Section 6.40).

12.3.2 Proxying Responses

A proxy only processes a response if the topmost Via field matches one of its addresses. A response with a non-matching top Via field MUST be dropped.

12.3.3 Stateless Proxy: Proxying Responses

A stateless proxy removes its own Via field, and checks the address in the next Via field. In the case of UDP, the response is sent to the address listed in the "maddr" tag if present, otherwise to the "received" tag if present, and finally to the address in the "sent-by" field. A proxy MUST remain stateful when handling requests received via TCP.

A stateless proxy MUST NOT generate its own provisional responses.

12.3.4 Stateful Proxy: Receiving Requests

When a stateful proxy receives a request, it checks the To, From (including tags), Call-ID and CSeq against existing request records. If the tuple exists, the request is a retransmission. The provisional or final response sent previously is retransmitted, as per the server state machine. If the tuple does not exist, the request corresponds to a new transaction, and the request should be proxied.

A stateful proxy server MAY generate its own provisional (1xx) responses.

12.3.5 Stateful Proxy: Receiving ACKs

When an ACK request is received, it is either processed locally or proxied. To make this determination, the To, From, CSeq and Call-ID fields are compared against those in previous requests. If there is no match, the ACK request is proxied as if it were an INVITE request. If there is a match, and if the server had ever sent a 200 response upstream, the ACK is proxied. If the server had never sent any responses upstream, the ACK is also proxied. If the server had sent a 3xx, 4xx, 5xx or 6xx response, but no 2xx response, the ACK is processed locally if the tag in the To field of the ACK matches the tag sent by the proxy in the response.

12.3.6 Stateful Proxy: Receiving Responses

When a proxy server receives a response that has passed the Via checks, the proxy server checks the To (without the tag), From (including the tag), Call-ID and CSeq against values seen in previous requests. If there is no match, the response is forwarded upstream to the address listed in the Via field. If there is a match, the "branch" tag in the Via field is examined. If it matches a known branch identifier, the response is for the given branch, and processed by the virtual client for the given branch. Otherwise, the response is dropped.

A stateful proxy should obey the rules in Section 12.4 to determine if the response should be proxied upstream. If it is to be proxied, the same rules for stateless proxies above are followed, with the following addition for TCP. If a request was received via TCP (indicated by the protocol in the top Via header), the proxy checks to see if it has a connection currently open to that address. If so, the response is sent on that connection. Otherwise, a new TCP connection is opened to the address and port in the Via field, and the response is sent there. Note that this implies that a UAC or proxy MUST be prepared to receive responses on the incoming side of a TCP connection. Definitive non 200-class responses MUST be retransmitted by the proxy, even over a TCP connection.

12.3.7 Stateless, Non-Forking Proxy

Proxies in this category issue at most a single unicast request for each incoming SIP request, that is, they do not "fork" requests. However, servers MAY choose to always operate in a mode that allows issuing of several requests, as described in Section 12.4.

The server can forward the request and any responses. It does not have to maintain any state for the SIP transaction. Reliability is assured by the next redirect or stateful proxy server in the server chain.

A proxy server SHOULD cache the result of any address translations and the response to speed forwarding of retransmissions. After the cache entry has been expired, the server cannot tell whether an incoming request is actually a retransmission of an older request. The server will treat it as a new request and commence another search.

12.4 Forking Proxy

The server MUST respond to the request immediately with a 100 (Trying) response.

Successful responses to an INVITE request MAY contain a Contact header field so that the following ACK or BYE bypasses the proxy search mechanism. If the proxy requires future requests to be routed through it, it adds a Record-Route header to the request (Section 6.29).

The following C-code describes the behavior of a proxy server issuing several requests in response to an incoming INVITE request. The function request(r, a, b) sends a SIP request of type r to address a, with branch id b. await_response() waits until a response is received and returns the response. close(a) closes the TCP connection to client with address a. response(r) sends a response to the client. ismulticast() returns 1 if the location is a multicast address and zero otherwise. The variable timeleft indicates the amount of time left until the maximum response time has expired. The variable recurse indicates whether the server will recursively try addresses returned through a 3xx response. A server MAY decide to recursively try only certain addresses, e.g., those which are within the same domain as the proxy server. Thus, an initial multicast request can trigger additional unicast requests.

```
/* request type */
typedef enum {INVITE, ACK, BYE, OPTIONS, CANCEL, REGISTER} Method;

process_request(Method R, int N, address_t address[])
{
  struct {
    int branch;         /* branch id */
    int done;           /* has responded */
  } outgoing[];
  int done[];           /* address has responded */
  char *location[];     /* list of locations */
  int heard = 0;        /* number of sites heard from */
  int class;            /* class of status code */
  int timeleft = 120;   /* sample timeout value */
  int loc = 0;          /* number of locations */
  struct {              /* response */
    int status;         /* response: CANCEL=-1 */
    int locations;      /* number of redirect locations */
    char *location[];   /* redirect locations */
    address_t a;        /* address of respondent */
    int branch;         /* branch identifier */
  } r, best;            /* response, best response */
  int i;

  best.status = 1000;
  for (i = 0; i < N; i++) {
```

```
          request(R, address[i], i);
          outgoing[i].done = 0;
          outgoing[i].branch = i;
        }

        while (timeleft > 0 && heard < N) {
          r = await_response();
          class = r.status / 100;

          /* If final response, mark branch as done. */
          if (class >= 2) {
            heard++;
            for (i = 0; i < N; i++) {
              if (r.branch == outgoing[i].branch) {
                outgoing[i].done = 1;
                break;
              }
            }
          }
          /* CANCEL: respond, fork and wait for responses */
          else if (class < 0) {
            best.status = 200;
            response(best);
            for (i = 0; i < N; i++) {
              if (!outgoing[i].done)
                request(CANCEL, address[i], outgoing[i].branch);
            }
            best.status = -1;
          }

          /* Send an ACK */

          if (class != 2) {
            if (R == INVITE) request(ACK, r.a, r.branch);
          }


          if (class == 2) {
            if (r.status < best.status) best = r;
            break;
          }
          else if (class == 3) {
            /* A server MAY optionally recurse.  The server MUST check
             * whether it has tried this location before and whether
             * the location is part of the Via path of the incoming
             * request.  This check is omitted here for brevity.
             * Multicast locations MUST NOT be returned to the client if
             * the server is not recursing.
```

```
           */
          if (recurse) {
            multicast = 0;
            N += r.locations;
            for (i = 0; i < r.locations; i++) {
              request(R, r.location[i]);
            }
          } else if (!ismulticast(r.location)) {
            best = r;
          }
        }
        else if (class == 4) {
          if (best.status >= 400) best = r;
        }
        else if (class == 5) {
          if (best.status >= 500) best = r;
        }
        else if (class == 6) {
          best = r;
          break;
        }
      }

      /* We haven't heard anything useful from anybody. */
      if (best.status == 1000) {
        best.status = 404;
      }
      if (best.status/100 != 3) loc = 0;
      response(best);
    }
```

Responses are processed as follows. The process completes (and state can be freed) when all requests have been answered by final status responses (for unicast) or 60 seconds have elapsed (for multicast). A proxy MAY send a CANCEL to all branches and return a 408 (Timeout) to the client after 60 seconds or more.

1xx: The proxy MAY forward the response upstream towards the client.

2xx: The proxy MUST forward the response upstream towards the client, without sending an ACK downstream. After receiving a 2xx, the server MAY terminate all other pending requests by sending a CANCEL request and closing the TCP connection, if applicable. (Terminating pending requests is advisable as searches consume resources. Also, INVITE requests could "ring" on a number of workstations if the callee is currently logged in more than once.)

3xx: The proxy MUST send an ACK and MAY recurse on the listed Contact addresses. Otherwise, the lowest-numbered response is returned if there were no 2xx responses.

Location lists are not merged as that would prevent forwarding of authenticated responses. Also, responses can have message bodies, so that merging is not feasible.

4xx, 5xx: The proxy MUST send an ACK and remember the response if it has a lower status code than any previous 4xx and 5xx responses. On completion, the lowest-numbered response is returned if there were no 2xx or 3xx responses.

6xx: The proxy MUST forward the response to the client and send an ACK. Other pending requests MAY be terminated with CANCEL as described for 2xx responses.

A proxy server forwards any response for Call-IDs for which it does not have a pending transaction according to the response's Via header. User agent servers respond to BYE requests for unknown call legs with status code 481 (Transaction Does Not Exist); they drop ACK requests with unknown call legs silently.

Special considerations apply for choosing forwarding destinations for ACK and BYE requests. In most cases, these requests will bypass proxies and reach the desired party directly, keeping proxies from having to make forwarding decisions.

A proxy MAY maintain call state for a period of its choosing. If a proxy still has list of destinations that it forwarded the last INVITE to, it SHOULD direct ACK requests only to those downstream servers.

13 Security Considerations

13.1 Confidentiality and Privacy: Encryption

13.1.1 End-to-End Encryption

SIP requests and responses can contain sensitive information about the communication patterns and communication content of individuals. The SIP message body MAY also contain encryption keys for the session itself. SIP supports three complementary forms of encryption to protect privacy:

- End-to-end encryption of the SIP message body and certain sensitive header fields;

- hop-by-hop encryption to prevent eavesdropping that tracks who is calling whom;

- hop-by-hop encryption of Via fields to hide the route a request has taken.

Not all of the SIP request or response can be encrypted end-to-end because header fields such as To and Via need to be visible to proxies so that the SIP request can be routed correctly. Hop-by-hop encryption encrypts the entire SIP request or response on the wire so that packet sniffers or other eavesdroppers cannot see who is calling whom. Hop-by-hop encryption can also encrypt requests and responses that have been end-to-end encrypted. Note that proxies can still see who is calling whom, and this information is also deducible by performing a network traffic analysis, so this provides a very limited but still worthwhile degree of protection.

SIP Via fields are used to route a response back along the path taken by the request and to prevent infinite request loops. However, the information given by them can also provide useful information to an attacker. Section 6.22 describes how a sender can request that Via fields be encrypted by cooperating proxies without compromising the purpose of the Via field.

End-to-end encryption relies on keys shared by the two user agents involved in the request. Typically, the message is sent encrypted with the public key of the recipient, so that only that recipient can read the message. All implementations SHOULD support PGP-based encryption [33] and MAY implement other schemes.

A SIP request (or response) is end-to-end encrypted by splitting the message to be sent into a part to be encrypted and a short header that will remain in the clear. Some parts of the SIP message, namely the request line, the response line and certain header fields marked with "n" in the "enc." column in Table 4 and 5 need to be read and returned by proxies and thus MUST NOT be encrypted end-to-end. Possibly sensitive information that needs to be made available as plaintext include destination address (To) and the forwarding path (Via) of the call. The Authorization header field MUST remain in the clear if it contains a digital signature as the signature is generated after encryption, but MAY be encrypted if it contains "basic" or "digest" authentication. The From header field SHOULD normally remain in the clear, but MAY be encrypted if required, in which case some proxies MAY return a 401 (Unauthorized) status if they require a From field.

Other header fields MAY be encrypted or MAY travel in the clear as desired by the sender. The Subject, Allow and Content-Type header fields will typically be encrypted. The Accept, Accept-Language, Date, Expires, Priority, Require, Call-ID, Cseq, and Timestamp header fields will remain in the clear.

All fields that will remain in the clear MUST precede those that will be encrypted. The message is encrypted starting with the first character of the first header field that will be encrypted and continuing through to the end of the message body. If no header fields are to be encrypted, encrypting starts with the second CRLF pair after the last header field, as shown below. Carriage return and line feed characters have been made visible as "$", and the encrypted part of the message is outlined.

```
  INVITE sip:watson@boston.bell-telephone.com SIP/2.0$
  Via: SIP/2.0/UDP 169.130.12.5$
  To: T. A. Watson <sip:watson@bell-telephone.com>$
  From: A. Bell <sip:a.g.bell@bell-telephone.com>$
  Encryption: PGP version=5.0$
  Content-Length: 224$
  Call-ID: 187602141351@worcester.bell-telephone.com$
  CSeq: 488$
  $
**************************************************
* Subject: Mr. Watson, come here.$               *
* Content-Type: application/sdp$                 *
* $                                              *
* v=0$                                           *
* o=bell 53655765 2353687637 IN IP4 128.3.4.5$   *
* c=IN IP4 135.180.144.94$                       *
* m=audio 3456 RTP/AVP 0 3 4 5$                  *
**************************************************
```

An Encryption header field MUST be added to indicate the encryption mechanism used. A Content-Length field is added that indicates the length of the encrypted body. The encrypted body is preceded by a blank line as a normal SIP message body would be.

Upon receipt by the called user agent possessing the correct decryption key, the message body as indicated by the Content-Length field is decrypted, and the now-decrypted body is appended to the clear-text header fields. There is no need for an additional Content-Length header field within the encrypted body because the length of the actual message body is unambiguous after decryption.

Had no SIP header fields required encryption, the message would have been as below. Note that the encrypted body MUST then include a blank line (start with CRLF) to disambiguate between any possible SIP header fields that might have been present and the SIP message body.

```
  INVITE sip:watson@boston.bell-telephone.com SIP/2.0$
  Via: SIP/2.0/UDP 169.130.12.5$
  To: T. A. Watson <sip:watson@bell-telephone.com>$
  From: A. Bell <a.g.bell@bell-telephone.com>$
  Encryption: PGP version=5.0$
  Content-Type: application/sdp$
  Content-Length: 107$
  $
**********************************************
* $                                          *
* v=0$                                       *
* o=bell 53655765 2353687637 IN IP4 128.3.4.5$  *
* c=IN IP4 135.180.144.94$                   *
* m=audio 3456 RTP/AVP 0 3 4 5$              *
**********************************************
```

13.1.2 Privacy of SIP Responses

SIP requests can be sent securely using end-to-end encryption and authentication to a called user agent that sends an insecure response. This is allowed by the SIP security model, but is not a good idea. However, unless the correct behavior is explicit, it would not always be possible for the called user agent to infer what a reasonable behavior was. Thus when end-to-end encryption is used by the request originator, the encryption key to be used for the response SHOULD be specified in the request. If this were not done, it might be possible for the called user agent to incorrectly infer an appropriate key to use in the response. Thus, to prevent key-guessing becoming an acceptable strategy, we specify that a called user agent receiving a request that does not specify a key to be used for the response SHOULD send that response unencrypted.

Any SIP header fields that were encrypted in a request SHOULD also be encrypted in an encrypted response. Contact response fields MAY be encrypted if the information they contain is sensitive, or MAY be left in the clear to permit proxies more scope for localized searches.

13.1.3 Encryption by Proxies

Normally, proxies are not allowed to alter end-to-end header fields and message bodies. Proxies MAY, however, encrypt an unsigned request or response with the key of the call recipient.

Proxies need to encrypt a SIP request if the end system cannot perform encryption or to enforce organizational security policies.

13.1.4 Hop-by-Hop Encryption

SIP requests and responses MAY also be protected by security mechanisms at the transport or network layer. No particular mechanism is defined or recommended here. Two possibilities are IPSEC [34] or TLS [35]. The use of a particular mechanism will generally need to be specified out of band, through manual configuration, for example.

13.1.5 Via field encryption

When Via header fields are to be hidden, a proxy that receives a request containing an appropriate "Hide: hop" header field (as specified in section 6.22) SHOULD encrypt the header field. As only the proxy that encrypts the field will decrypt it, the algorithm chosen is entirely up to the proxy implementor. Two methods satisfy these requirements:

- The server keeps a cache of Via header fields and the associated To header field, and replaces the Via header field with an index into the cache. On the reverse path, take the Via header field from the cache rather than the message.

This is insufficient to prevent message looping, and so an additional ID MUST be added so that the proxy can detect loops. This SHOULD NOT normally be the address of the proxy as the goal is to hide the route, so instead a sufficiently large random number SHOULD be used by the proxy and maintained in the cache.

It is possible for replies to get directed to the wrong originator if the cache entry gets reused, so great care needs to be taken to ensure this does not happen.

- The server MAY use a secret key to encrypt the Via field, a timestamp and an appropriate checksum in any such message with the same secret key. The checksum is needed to detect whether successful decoding has occurred, and the timestamp is

required to prevent possible replay attacks and to ensure that no two requests from the same previous hop have the same encrypted Via field. This is the preferred solution.

13.2 Message Integrity and Access Control: Authentication

Protective measures need to be taken to prevent an active attacker from modifying and replaying SIP requests and responses. The same cryptographic measures that are used to ensure the authenticity of the SIP message also serve to authenticate the originator of the message. However, the "basic" and "digest" authentication mechanism offer authentication only, without message integrity.

Transport-layer or network-layer authentication MAY be used for hop-by-hop authentication. SIP also extends the HTTP WWW-Authenticate (Section 6.42) and Authorization (Section 6.11) header field and their Proxy counterparts to include cryptographically strong signatures. SIP also supports the HTTP "basic" and "digest" schemes (see Section 14) and other HTTP authentication schemes to be defined that offer a rudimentary mechanism of ascertaining the identity of the caller.

Since SIP requests are often sent to parties with which no prior communication relationship has existed, we do not specify authentication based on shared secrets.

SIP requests MAY be authenticated using the Authorization header field to include a digital signature of certain header fields, the request method and version number and the payload, none of which are modified between client and called user agent. The Authorization header field is used in requests to authenticate the request originator end-to-end to proxies and the called user agent, and in responses to authenticate the called user agent or proxies returning their own failure codes. If required, hop-by-hop authentication can be provided, for example, by the IPSEC Authentication Header.

SIP does not dictate which digital signature scheme is used for authentication, but does define how to provide authentication using PGP in Section 15. As indicated above, SIP implementations MAY also use "basic" and "digest" authentication and other authentication mechanisms defined for HTTP. Note that "basic" authentication has severe security limitations. The following does not apply to these schemes.

To cryptographically sign a SIP request, the order of the SIP header fields is important. When an Authorization header field is present, it indicates that all header fields following the Authorization

header field have been included in the signature. Therefore, hop-by-hop header fields which MUST or SHOULD be modified by proxies MUST precede the Authorization header field as they will generally be modified or added-to by proxy servers. Hop-by-hop header fields which MAY be modified by a proxy MAY appear before or after the Authorization header. When they appear before, they MAY be modified by a proxy. When they appear after, they MUST NOT be modified by a proxy. To sign a request, a client constructs a message from the request method (in upper case) followed, without LWS, by the SIP version number, followed, again without LWS, by the request headers to be signed and the message body. The message thus constructed is then signed.

For example, if the SIP request is to be:

```
INVITE sip:watson@boston.bell-telephone.com SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
Authorization: PGP version=5.0, signature=...
From: A. Bell <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
c=IN IP4 135.180.144.94
m=audio 3456 RTP/AVP 0 3 4 5
```

Then the data block that is signed is:

```
INVITESIP/2.0From: A. Bell <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
c=IN IP4 135.180.144.94
m=audio 3456 RTP/AVP 0 3 4 5
```

Clients wishing to authenticate requests MUST construct the portion of the message below the Authorization header using a canonical form. This allows a proxy to parse the message, take it apart, and reconstruct it, without causing an authentication failure due to extra white space, for example. Canonical form consists of the following rules:

- No short form header fields
- Header field names are capitalized as shown in this document
- No white space between the header name and the colon
- A single space after the colon
- Line termination with a CRLF
- No line folding
- No comma separated lists of header values; each must appear as a separate header
- Only a single SP between tokens, between tokens and quoted strings, and between quoted strings; no SP after last token or quoted string
- No LWS between tokens and separators, except as described above for after the colon in header fields

Note that if a message is encrypted and authenticated using a digital signature, when the message is generated encryption is performed before the digital signature is generated. On receipt, the digital signature is checked before decryption.

A client MAY require that a server sign its response by including a Require: org.ietf.sip.signed-response request header field. The client indicates the desired authentication method via the WWW-Authenticate header.

The correct behavior in handling unauthenticated responses to a request that requires authenticated responses is described in section 13.2.1.

13.2.1 Trusting responses

There is the possibility that an eavesdropper listens to requests and then injects unauthenticated responses that terminate, redirect or otherwise interfere with a call. (Even encrypted requests contain enough information to fake a response.)

Clients need to be particularly careful with 3xx redirection responses. Thus a client receiving, for example, a 301 (Moved Permanently) which was not authenticated when the public key of the called user agent is known to the client, and authentication was requested in the request SHOULD be treated as suspicious. The correct behavior in such a case would be for the called-user to form a dated response containing the Contact field to be used, to sign it, and give this signed stub response to the proxy that will provide the redirection. Thus the response can be authenticated correctly. A client SHOULD NOT automatically redirect such a request to the new location without alerting the user to the authentication failure before doing so.

Another problem might be responses such as 6xx failure responses which would simply terminate a search, or "4xx" and "5xx" response failures.

If TCP is being used, a proxy SHOULD treat 4xx and 5xx responses as valid, as they will not terminate a search. However, fake 6xx responses from a rogue proxy terminate a search incorrectly. 6xx responses SHOULD be authenticated if requested by the client, and failure to do so SHOULD cause such a client to ignore the 6xx response and continue a search.

With UDP, the same problem with 6xx responses exists, but also an active eavesdropper can generate 4xx and 5xx responses that might cause a proxy or client to believe a failure occurred when in fact it did not. Typically 4xx and 5xx responses will not be signed by the called user agent, and so there is no simple way to detect these rogue responses. This problem is best prevented by using hop-by-hop encryption of the SIP request, which removes any additional problems that UDP might have over TCP.

These attacks are prevented by having the client require response authentication and dropping unauthenticated responses. A server user agent that cannot perform response authentication responds using the normal Require response of 420 (Bad Extension).

13.3 Callee Privacy

User location and SIP-initiated calls can violate a callee's privacy. An implementation SHOULD be able to restrict, on a per-user basis, what kind of location and availability information is given out to certain classes of callers.

13.4 Known Security Problems

With either TCP or UDP, a denial of service attack exists by a rogue proxy sending 6xx responses. Although a client SHOULD choose to ignore such responses if it requested authentication, a proxy cannot do so. It is obliged to forward the 6xx response back to the client. The client can then ignore the response, but if it repeats the request it will probably reach the same rogue proxy again, and the process will repeat.

14 SIP Authentication using HTTP Basic and Digest Schemes

SIP implementations MAY use HTTP's basic and digest authentication mechanisms to provide a rudimentary form of security. This section overviews usage of these mechanisms in SIP. The basic operation is almost completely identical to that for HTTP [36]. This section outlines this operation, pointing to [36] for details, and noting the differences when used in SIP.

14.1 Framework

The framework for SIP authentication parallels that for HTTP [36]. In particular, the BNF for auth-scheme, auth-param, challenge, realm, realm-value, and credentials is identical. The 401 response is used by user agent servers in SIP to challenge the authorization of a user agent client. Additionally, registrars and redirect servers MAY make use of 401 responses for authorization, but proxies MUST NOT, and instead MAY use the 407 response. The requirements for inclusion of the Proxy-Authenticate, Proxy-Authorization, WWW-Authenticate, and Authorization in the various messages is identical to [36].

Since SIP does not have the concept of a canonical root URL, the notion of protections spaces are interpreted differently for SIP. The realm is a protection domain for all SIP URIs with the same value for the userinfo, host and port part of the SIP Request-URI. For example:

```
INVITE sip:alice.wonderland@example.com SIP/2.0
WWW-Authenticate:  Basic realm="business"
```

and

```
INVITE sip:aw@example.com SIP/2.0
WWW-Authenticate: Basic realm="business"
```

define different protection realms according to this rule.

When a UAC resubmits a request with its credentials after receiving a 401 or 407 response, it MUST increment the CSeq header field as it would normally do when sending an updated request.

14.2 Basic Authentication

The rules for basic authentication follow those defined in [36], but with the words "origin server" replaced with "user agent server, redirect server , or registrar".

Since SIP URIs are not hierarchical, the paragraph in [36] that states that "all paths at or deeper than the depth of the last symbolic element in the path field of the Request-URI also are within the protection space specified by the Basic realm value of the current challenge" does not apply for SIP. SIP clients MAY preemptively send the corresponding Authorization header with requests for SIP URIs within the same protection realm (as defined above) without receipt of another challenge from the server.

14.3 Digest Authentication

The rules for digest authentication follow those defined in [36], with "HTTP 1.1" replaced by "SIP/2.0" in addition to the following differences:

1. The URI included in the challenge has the following BNF:

   ```
   URI  =  SIP-URL
   ```

2. The BNF for digest-uri-value is:

   ```
   digest-uri-value  =  Request-URI ; a defined in Section
   4.3
   ```

3. The example procedure for choosing a nonce based on Etag does not work for SIP.

4. The Authentication-Info and Proxy-Authentication-Info fields are not used in SIP.

5. The text in [36] regarding cache operation does not apply to SIP.

6. [36] requires that a server check that the URI in the request line, and the URI included in the Authorization header, point to the same resource. In a SIP context, these two URI's may actually refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the request-uri in the Authorization header corresponds to a user that the server is willing to accept forwarded or direct calls for.

14.4 Proxy-Authentication

The use of the Proxy-Authentication and Proxy-Authorization parallel that as described in [36], with one difference. Proxies MUST NOT add the Proxy-Authorization header. 407 responses MUST be forwarded upstream towards the client following the procedures for any other response. It is the client's responsibility to add the Proxy-Authorization header containing credentials for the proxy which has asked for authentication.

If a proxy were to resubmit a request with a Proxy-Authorization header field, it would need to increment the CSeq in the new request. However, this would mean that the UAC which submitted the original request would discard a response from the UAS, as the CSeq value would be different.

See sections 6.26 and 6.27 for additional information on usage of these fields as they apply to SIP.

15 SIP Security Using PGP

15.1 PGP Authentication Scheme

The "pgp" authentication scheme is based on the model that the client authenticates itself with a request signed with the client's private key. The server can then ascertain the origin of the request if it has access to the public key, preferably signed by a trusted third party.

15.1.1 The WWW-Authenticate Response Header

```
WWW-Authenticate =  "WWW-Authenticate" ":" "pgp" pgp-challenge
pgp-challenge    =  * (";" pgp-params )
pgp-params       =  realm | pgp-version | pgp-algorithm | nonce
realm            =  "realm" "=" realm-value
realm-value      =  quoted-string
pgp-version      =  "version" "="
                     <"> digit *( "." digit ) *letter <">
pgp-algorithm    =  "algorithm" "=" ( "md5" | "sha1" | token )
nonce            =  "nonce" "=" nonce-value
nonce-value      =  quoted-string
```

The meanings of the values of the parameters used above are as follows:

realm: A string to be displayed to users so they know which identity to use. This string SHOULD contain at least the name of the host performing the authentication and MAY additionally indicate the collection of users who might have access. An example might be " Users with call-out privileges ".

pgp-algorithm: The value of this parameter indicates the PGP message integrity check (MIC) to be used to produce the signature. If this not present it is assumed to be "md5". The currently defined values are "md5" for the MD5 checksum, and "sha1" for the SHA.1 algorithm.

pgp-version: The version of PGP that the client MUST use. Common values are "2.6.2" and "5.0". The default is 5.0.

nonce: A server-specified data string which should be uniquely generated each time a 401 response is made. It is RECOMMENDED that this string be base64 or hexadecimal data. Specifically, since the string is passed in the header lines as a quoted string, the double-quote character is not allowed. The contents of the nonce are implementation dependent. The quality of the implementation depends on a good choice. Since the nonce is used only to prevent replay attacks and is signed, a time stamp in units convenient to the server is sufficient.

Replay attacks within the duration of the call setup are of limited interest, so that timestamps with a resolution of a few seconds are often should be sufficient. In that case, the server does not have to keep a record of the nonces.

Example:

```
WWW-Authenticate: pgp ;version="5.0"
  ;realm="Your Startrek identity, please" ;algorithm=md5
  ;nonce="913082051"
```

15.1.2 The Authorization Request Header

The client is expected to retry the request, passing an Authorization header line, which is defined as follows.

```
Authorization  =  "Authorization" ":" "pgp" *( ";" pgp-response )
pgp-response   =  realm | pgp-version | pgp-signature
                  | signed-by | nonce
pgp-signature  =  "signature" "=" quoted-string
signed-by      =  "signed-by" "=" <"> URI <">
```

The client MUST increment the CSeq header before resubmitting the request. The signature MUST correspond to the From header of the request unless the signed-by parameter is provided.

pgp-signature: The PGP ASCII-armored signature [33], as it appears between the "BEGIN PGP MESSAGE" and "END PGP MESSAGE" delimiters, without the version indication. The signature is included without any linebreaks.

The signature is computed across the nonce (if present), request method, request version and header fields following the Authorization header and the message body, in the same order as they appear in the message. The request method and version are prepended to the header fields without any white space. The signature is computed across the headers as sent, and the terminating CRLF. The CRLF following the Authorization header is NOT included in the signature.

A server MAY be configured not to generate nonces only if replay attacks are not a concern.

Not generating nonces avoids the additional set of request, 401 response and possibly ACK messages and reduces delay by one round-trip time.

Using the ASCII-armored version is about 25% less space-efficient than including the binary signature, but it is significantly easier for the receiver to piece together. Versions of the PGP program always include the full (compressed) signed text in their output unless ASCII-armored mode (-sta) is specified. Typical signatures are about 200 bytes long. -- The PGP signature mechanism allows the client to simply pass the request to an external PGP program. This relies on the requirement that proxy servers are not allowed to reorder or change header fields.

realm: The realm is copied from the corresponding WWW-Authenticate header field parameter.

signed-by: If and only if the request was not signed by the entity listed in the From header, the signed-by header indicates the name of the signing entity, expressed as a URI.

Receivers of signed SIP messages SHOULD discard any end-to-end header fields above the Authorization header, as they may have been maliciously added en route by a proxy.

Example:

```
Authorization: pgp version="5.0"
  ;realm="Your Startrek identity, please"
  ;nonce="913082051"
  ;signature="iQB1AwUBNNJiUaYBnHmiiQh1AQFYsgL/Wt3dk6TWK81/b0gcNDf
  VAUGU4rhEBW972IPxFSOZ94L1qhCLInTPaqhHFw1cb3lB01rA0RhpV4t5yCdUt
  SRYBSkOK29o5e1KlFeW23EzYPVUm2TlDAhbcjbMdfC+KLFX
  =aIrx"
```

15.2 PGP Encryption Scheme

The PGP encryption scheme uses the following syntax:

```
Encryption    =  "Encryption" ":" "pgp" pgp-eparams
pgp-eparams   =  1# ( pgp-version | pgp-encoding )
pgp-encoding  =  "encoding" "=" "ascii" | token
```

encoding: Describes the encoding or "armor" used by PGP. The value "ascii" refers to the standard PGP ASCII armor, without the lines containing "BEGIN PGP MESSAGE" and "END PGP MESSAGE" and without the version identifier. By default, the encrypted part is included as binary.

Example:

Encryption: pgp version="2.6.2", encoding="ascii"

15.3 Response-Key Header Field for PGP

```
Response-Key  =  "Response-Key" ":" "pgp" pgp-eparams
pgp-eparams   =  1# ( pgp-version | pgp-encoding | pgp-key)
pgp-key       =  "key" "=" quoted-string
```

If ASCII encoding has been requested via the encoding parameter, the key parameter contains the user's public key as extracted from the pgp key ring with the "pgp -kxa user ".

Example:

```
Response-Key: pgp version="2.6.2", encoding="ascii",
  key="mQBtAzNWHNYAAAEDAL7QvAdK2utY05wuUG+ItYK5tCF8HNJM60sU4rLaV+eUnkMk
  mOmJWtc2wXcZx1XaXb2lkydTQOesrUR75IwNXBuZXPEIMThEa5WLsT7VLme7njnx
  sE86SgWmAZx5ookIdQAFEbQxSGVubmluZyBTY2h1bHpyaW5uZSA8c2NodWx6cmlu
  bmVAY3MuY29sdW1iaWEuZWR1Pg==
  =+y19"
```

16 Examples

In the following examples, we often omit the message body and the corresponding Content-Length and Content-Type headers for brevity.

16.1 Registration

A user at host saturn.bell-tel.com registers on start-up, via multicast, with the local SIP server named bell-tel.com. In the example, the user agent on saturn expects to receive SIP requests on UDP port 3890.

```
C->S: REGISTER sip:bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 1 REGISTER
      Contact: <sip:watson@saturn.bell-tel.com:3890;transport=udp>
      Expires: 7200
```

The registration expires after two hours. Any future invitations for watson@bell-tel.com arriving at sip.bell-tel.com will now be redirected to watson@saturn.bell-tel.com, UDP port 3890.

If Watson wants to be reached elsewhere, say, an on-line service he uses while traveling, he updates his reservation after first cancelling any existing locations:

```
C->S: REGISTER sip:bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 2 REGISTER
      Contact: *
      Expires: 0

C->S: REGISTER sip:bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 3 REGISTER
      Contact: sip:tawatson@example.com
```

Now, the server will forward any request for Watson to the server at example.com, using the Request-URI tawatson@example.com. For the server at example.com to reach Watson, he will need to send a REGISTER there, or inform the server of his current location through some other means.

It is possible to use third-party registration. Here, the secretary jon.diligent registers his boss, T. Watson:

```
C->S: REGISTER sip:bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP pluto.bell-tel.com
      From: sip:jon.diligent@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 17320@pluto.bell-tel.com
      CSeq: 1 REGISTER
      Contact: sip:tawatson@example.com
```

The request could be sent to either the registrar at bell-tel.com or the server at example.com. In the latter case, the server at example.com would proxy the request to the address indicated in the Request-URI. Then, Max-Forwards header could be used to restrict the registration to that server.

16.2 Invitation to a Multicast Conference

The first example invites schooler@vlsi.cs.caltech.edu to a multicast session. All examples use the Session Description Protocol (SDP) (RFC 2327 [6]) as the session description format.

16.2.1 Request

```
C->S: INVITE sip:schooler@cs.caltech.edu SIP/2.0
      Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
        ;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP north.east.isi.edu
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu>
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 INVITE
      Subject: SIP will be discussed, too
      Content-Type: application/sdp
      Content-Length: 187

      v=0
      o=user1 53655765 2353687637 IN IP4 128.3.4.5
      s=Mbone Audio
      i=Discussion of Mbone Engineering Issues
      e=mbone@somewhere.com
      c=IN IP4 224.2.0.1/127
      t=0 0
      m=audio 3456 RTP/AVP 0
```

The From request header above states that the request was initiated by mjh@isi.edu and addressed to schooler@caltech.edu (From header fields). The Via fields list the hosts along the path from invitation initiator (the last element of the list) towards the callee. In the example above, the message was last multicast to the administratively scoped group 239.128.16.254 with a ttl of 16 from the host csvax.cs.caltech.edu. The second Via header field indicates that it was originally sent from the host north.east.isi.edu. The Request-URI indicates that the request is currently being being addressed to schooler@cs.caltech.edu, the local address that csvax looked up for the callee.

In this case, the session description is using the Session Description Protocol (SDP), as stated in the Content-Type header.

The header is terminated by an empty line and is followed by a message body containing the session description.

16.2.2 Response

The called user agent, directly or indirectly through proxy servers, indicates that it is alerting ("ringing") the called party:

```
S->C: SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
        ;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP north.east.isi.edu
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu> ;tag=9883472
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 INVITE
```

A sample response to the invitation is given below. The first line of the response states the SIP version number, that it is a 200 (OK) response, which means the request was successful. The Via headers are taken from the request, and entries are removed hop by hop as the response retraces the path of the request. A new authentication field MAY be added by the invited user's agent if required. The Call-ID is taken directly from the original request, along with the remaining fields of the request message. The original sense of From field is preserved (i.e., it is the session initiator).

In addition, the Contact header gives details of the host where the user was located, or alternatively the relevant proxy contact point which should be reachable from the caller's host.

```
S->C: SIP/2.0 200 OK
      Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
        ;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP north.east.isi.edu
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu> ;tag=9883472
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 INVITE
      Contact: sip:es@jove.cs.caltech.edu
```

The caller confirms the invitation by sending an ACK request to the location named in the Contact header:

```
C->S: ACK sip:es@jove.cs.caltech.edu SIP/2.0
      Via: SIP/2.0/UDP north.east.isi.edu
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu> ;tag=9883472
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 ACK
```

16.3 Two-party Call

For two-party Internet phone calls, the response must contain a description of where to send the data. In the example below, Bell calls Watson. Bell indicates that he can receive RTP audio codings 0 (PCMU), 3 (GSM), 4 (G.723) and 5 (DVI4).

```
C->S: INVITE sip:watson@boston.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Subject: Mr. Watson, come here.
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=bell 53655765 2353687637 IN IP4 128.3.4.5
      s=Mr. Watson, come here.
      c=IN IP4 kton.bell-tel.com
      m=audio 3456 RTP/AVP 0 3 4 5
```

```
   S->C: SIP/2.0 100 Trying
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Content-Length: 0

   S->C: SIP/2.0 180 Ringing
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Content-Length: 0

   S->C: SIP/2.0 182 Queued, 2 callers ahead
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Content-Length: 0

   S->C: SIP/2.0 182 Queued, 1 caller ahead
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Content-Length: 0

   S->C: SIP/2.0 200 OK
         Via: SIP/2.0/UDP kton.bell-tel.com
         From: A. Bell <sip:a.g.bell@bell-tel.com>
         To: <sip:watson@bell-tel.com> ;tag=37462311
         Call-ID: 3298420296@kton.bell-tel.com
         CSeq: 1 INVITE
         Contact: sip:watson@boston.bell-tel.com
         Content-Type: application/sdp
         Content-Length: ...

         v=0
         o=watson 4858949 4858949 IN IP4 192.1.2.3
         s=I'm on my way
         c=IN IP4 boston.bell-tel.com
         m=audio 5004 RTP/AVP 0 3
```

The example illustrates the use of informational status responses. Here, the reception of the call is confirmed immediately (100), then, possibly after some database mapping delay, the call rings (180) and is then queued, with periodic status updates.

Watson can only receive PCMU and GSM. Note that Watson's list of codecs may or may not be a subset of the one offered by Bell, as each party indicates the data types it is willing to receive. Watson will send audio data to port 3456 at c.bell-tel.com, Bell will send to port 5004 at boston.bell-tel.com.

By default, the media session is one RTP session. Watson will receive RTCP packets on port 5005, while Bell will receive them on port 3457.

Since the two sides have agreed on the set of media, Bell confirms the call without enclosing another session description:

```
C->S: ACK sip:watson@boston.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 ACK
```

16.4 Terminating a Call

To terminate a call, caller or callee can send a BYE request:

```
C->S: BYE sip:watson@boston.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. A. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 2 BYE
```

If the callee wants to abort the call, it simply reverses the To and From fields. Note that it is unlikely that a BYE from the callee will traverse the same proxies as the original INVITE.

16.5 Forking Proxy

In this example, Bell (a.g.bell@bell-tel.com) (C), currently seated at host c.bell-tel.com wants to call Watson (t.watson@ieee.org). At the time of the call, Watson is logged in at two workstations, t.watson@x.bell-tel.com (X) and watson@y.bell-tel.com (Y), and has registered with the IEEE proxy server (P) called sip.ieee.org. The IEEE server also has a registration for the home machine of Watson, at watson@h.bell-tel.com (H), as well as a permanent registration at watson@acm.org (A). For brevity, the examples omit the session description and Via header fields.

Bell's user agent sends the invitation to the SIP server for the ieee.org domain:

```
C->P: INVITE sip:t.watson@ieee.org SIP/2.0
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

The SIP server at ieee.org tries the four addresses in parallel. It sends the following message to the home machine:

```
P->H: INVITE sip:watson@h.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=1
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

This request immediately yields a 404 (Not Found) response, since Watson is not currently logged in at home:

```
H->P: SIP/2.0 404 Not Found
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=1
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>;tag=87454273
```

```
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

The proxy ACKs the response so that host H can stop retransmitting it:

```
P->H: ACK sip:watson@h.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=1
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>;tag=87454273
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 ACK
```

Also, P attempts to reach Watson through the ACM server:

```
P->A: INVITE sip:watson@acm.org SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=2
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

In parallel, the next attempt proceeds, with an INVITE to X and Y:

```
P->X: INVITE sip:t.watson@x.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=3
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE

P->Y: INVITE sip:watson@y.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=4
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

As it happens, both Watson at X and a colleague in the other lab at host Y hear the phones ringing and pick up. Both X and Y return 200s via the proxy to Bell.

```
X->P: SIP/2.0 200 OK
      Via:      SIP/2.0/UDP sip.ieee.org ;branch=3
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org> ;tag=192137601
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 INVITE
      Contact:  sip:t.watson@x.bell-tel.com

Y->P: SIP/2.0 200 OK
      Via:      SIP/2.0/UDP sip.ieee.org ;branch=4
      Via:      SIP/2.0/UDP c.bell-tel.com
      Contact:  sip:t.watson@y.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org> ;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 INVITE
```

Both responses are forwarded to Bell, using the Via information. At this point, the ACM server is still searching its database. P can now cancel this attempt:

```
P->A: CANCEL sip:watson@acm.org SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=2
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 CANCEL
```

The ACM server gladly stops its neural-network database search and responds with a 200. The 200 will not travel any further, since P is the last Via stop.

```
A->P: SIP/2.0 200 OK
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=2
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
```

```
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 CANCEL
```

Bell gets the two 200 responses from X and Y in short order. Bell's reaction now depends on his software. He can either send an ACK to both if human intelligence is needed to determine who he wants to talk to or he can automatically reject one of the two calls. Here, he acknowledges both, separately and directly to the final destination:

```
C->X: ACK sip:t.watson@x.bell-tel.com SIP/2.0
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>;tag=192137601
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 ACK

C->Y: ACK sip:watson@y.bell-tel.com SIP/2.0
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 ACK
```

After a brief discussion between Bell with X and Y, it becomes clear that Watson is at X. (Note that this is not a three-way call; only Bell can talk to X and Y, but X and Y cannot talk to each other.) Thus, Bell sends a BYE to Y, which is replied to:

```
C->Y: BYE sip:watson@y.bell-tel.com SIP/2.0
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     2 BYE

Y->C: SIP/2.0 200 OK
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     2 BYE
```

16.6 Redirects

Replies with status codes 301 (Moved Permanently) or 302 (Moved Temporarily) specify another location using the Contact field. Continuing our earlier example, the server P at ieee.org decides to redirect rather than proxy the request:

```
P->C: SIP/2.0 302 Moved temporarily
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>;tag=72538263
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: sip:watson@h.bell-tel.com,
                sip:watson@acm.org, sip:t.watson@x.bell-tel.com,
                sip:watson@y.bell-tel.com
      CSeq: 1 INVITE
```

As another example, assume Alice (A) wants to delegate her calls to Bob (B) while she is on vacation until July 29th, 1998. Any calls meant for her will reach Bob with Alice's To field, indicating to him what role he is to play. Charlie (C) calls Alice (A), whose server returns:

```
A->C: SIP/2.0 302 Moved temporarily
      From: Charlie <sip:charlie@caller.com>
      To: Alice <sip:alice@anywhere.com> ;tag=2332462
      Call-ID: 27182@caller.com
      Contact: sip:bob@anywhere.com
      Expires: Wed, 29 Jul 1998 9:00:00 GMT
      CSeq: 1 INVITE
```

Charlie then sends the following request to the SIP server of the anywhere.com domain. Note that the server at anywhere.com forwards the request to Bob based on the Request-URI.

```
C->B: INVITE sip:bob@anywhere.com SIP/2.0
      From: sip:charlie@caller.com
      To: sip:alice@anywhere.com
      Call-ID: 27182@caller.com
      CSeq: 2 INVITE
```

In the third redirection example, we assume that all outgoing requests are directed through a local firewall F at caller.com, with Charlie again inviting Alice:

```
C->F: INVITE sip:alice@anywhere.com SIP/2.0
      From: sip:charlie@caller.com
      To: Alice <sip:alice@anywhere.com>
      Call-ID: 27182@caller.com
      CSeq: 1 INVITE
```

The local firewall at caller.com happens to be overloaded and thus redirects the call from Charlie to a secondary server S:

```
F->C: SIP/2.0 302 Moved temporarily
      From: sip:charlie@caller.com
      To: Alice <sip:alice@anywhere.com>
      Call-ID: 27182@caller.com
      CSeq: 1 INVITE
      Contact: <sip:alice@anywhere.com:5080;maddr=spare.caller.com>
```

Based on this response, Charlie directs the same invitation to the secondary server spare.caller.com at port 5080, but maintains the same Request-URI as before:

```
C->S: INVITE sip:alice@anywhere.com SIP/2.0
      From: sip:charlie@caller.com
      To: Alice <sip:alice@anywhere.com>
      Call-ID: 27182@caller.com
      CSeq: 2 INVITE
```

16.7 Negotiation

An example of a 606 (Not Acceptable) response is:

```
S->C: SIP/2.0 606 Not Acceptable
      From: sip:mjh@isi.edu
      To: <sip:schooler@cs.caltech.edu> ;tag=7434264
      Call-ID: 14142@north.east.isi.edu
```

```
CSeq: 1 INVITE
Contact: sip:mjh@north.east.isi.edu
Warning: 370 "Insufficient bandwidth (only have ISDN)",
  305 "Incompatible media format",
  330 "Multicast not available"
Content-Type: application/sdp
Content-Length: 50

v=0
s=Let's talk
b=CT:128
c=IN IP4 north.east.isi.edu
m=audio 3456 RTP/AVP 5 0 7
m=video 2232 RTP/AVP 31
```

In this example, the original request specified a bandwidth that was higher than the access link could support, requested multicast, and requested a set of media encodings. The response states that only 128 kb/s is available and that (only) DVI, PCM or LPC audio could be supported in order of preference.

The response also states that multicast is not available. In such a case, it might be appropriate to set up a transcoding gateway and re-invite the user.

16.8 OPTIONS Request

A caller Alice can use an OPTIONS request to find out the capabilities of a potential callee Bob, without "ringing" the designated address. Bob returns a description indicating that he is capable of receiving audio encodings PCM Ulaw (payload type 0), 1016 (payload type 1), GSM (payload type 3), and SX7300/8000 (dynamic payload type 99), and video encodings H.261 (payload type 31) and H.263 (payload type 34).

```
C->S: OPTIONS sip:bob@example.com SIP/2.0
      From: Alice <sip:alice@anywhere.org>
      To: Bob <sip:bob@example.com>
      Call-ID: 6378@host.anywhere.org
      CSeq: 1 OPTIONS
      Accept: application/sdp

S->C: SIP/2.0 200 OK
      From: Alice <sip:alice@anywhere.org>
      To: Bob <sip:bob@example.com> ;tag=376364382
```

```
Call-ID: 6378@host.anywhere.org
Content-Length: 81
Content-Type: application/sdp

v=0
m=audio 0 RTP/AVP 0 1 3 99
m=video 0 RTP/AVP 31 34
a=rtpmap:99 SX7300/8000
```

A Minimal Implementation

A.1 Client

All clients MUST be able to generate the INVITE and ACK requests. Clients MUST generate and parse the Call-ID, Content-Length, Content-Type, CSeq, From and To headers. Clients MUST also parse the Require header. A minimal implementation MUST understand SDP (RFC 2327, [6]). It MUST be able to recognize the status code classes 1 through 6 and act accordingly.

The following capability sets build on top of the minimal implementation described in the previous paragraph. In general, each capability listed below builds on the ones above it:

Basic: A basic implementation adds support for the BYE method to allow the interruption of a pending call attempt. It includes a User-Agent header in its requests and indicates its preferred language in the Accept-Language header.

Redirection: To support call forwarding, a client needs to be able to understand the Contact header, but only the SIP-URL part, not the parameters.

Firewall-friendly: A firewall-friendly client understands the Route and Record-Route header fields and can be configured to use a local proxy for all outgoing requests.

Negotiation: A client MUST be able to request the OPTIONS method and understand the 380 (Alternative Service) status and the Contact parameters to participate in terminal and media negotiation. It SHOULD be able to parse the Warning response header to provide useful feedback to the caller.

Authentication: If a client wishes to invite callees that require caller authentication, it MUST be able to recognize the 401 (Unauthorized) status code, MUST be able to generate the Authorization request header and MUST understand the WWW-Authenticate response header.

If a client wishes to use proxies that require caller authentication, it MUST be able to recognize the 407 (Proxy Authentication Required) status code, MUST be able to generate the Proxy-Authorization request header and understand the Proxy-Authenticate response header.

A.2 Server

A minimally compliant server implementation MUST understand the INVITE, ACK, OPTIONS and BYE requests. A proxy server MUST also understand CANCEL. It MUST parse and generate, as appropriate, the Call-ID, Content-Length, Content-Type, CSeq, Expires, From, Max-Forwards, Require, To and Via headers. It MUST echo the CSeq and Timestamp headers in the response. It SHOULD include the Server header in its responses.

A.3 Header Processing

Table 6 lists the headers that different implementations support. UAC refers to a user-agent client (calling user agent), UAS to a user-agent server (called user-agent).

The fields in the table have the following meaning. Type is as in Table 4 and 5. "-" indicates the field is not meaningful to this system (although it might be generated by it). "m" indicates the field MUST be understood. "b" indicates the field SHOULD be understood by a Basic implementation. "r" indicates the field SHOULD be understood if the system claims to understand redirection. "a" indicates the field SHOULD be understood if the system claims to support authentication. "e" indicates the field SHOULD be understood if the system claims to support encryption. "o" indicates support of the field is purely optional. Headers whose support is optional for all implementations are not shown.

	type	UAC	proxy	UAS	registrar
Accept	R	-	o	m	m
Accept-Encoding	R	-	-	m	m
Accept-Language	R	-	b	b	b
Allow	405	o	-	-	-
Authorization	R	a	o	a	a
Call-ID	g	m	m	m	m
Content-Encoding	g	m	-	m	m
Content-Length	g	m	m	m	m
Content-Type	g	m	-	m	m
CSeq	g	m	m	m	m
Encryption	g	e	-	e	e
Expires	g	-	o	o	m
From	g	m	o	m	m
Hide	R	-	m	-	-
Contact	R	-	-	-	m
Contact	r	r	r	-	-
Max-Forwards	R	-	b	-	-
Proxy-Authenticate	407	a	-	-	-
Proxy-Authorization	R	-	a	-	-
Proxy-Require	R	-	m	-	-
Require	R	m	-	m	m
Response-Key	R	-	-	e	e
Route	R	-	m	-	-
Timestamp	g	o	o	m	m
To	g	m	m	m	m
Unsupported	r	b	b	-	-
User-Agent	g	b	-	b	-
Via	g	m	m	m	m
WWW-Authenticate	401	a	-	-	-

Table 6: Header Field Processing Requirements

B Usage of the Session Description Protocol (SDP)

This section describes the use of the Session Description Protocol (SDP) (RFC 2327 [6]).

B.1 Configuring Media Streams

The caller and callee align their media descriptions so that the nth media stream ("m=" line) in the caller's session description corresponds to the nth media stream in the callee's description.

All media descriptions SHOULD contain "a=rtpmap" mappings from RTP payload types to encodings.

This allows easier migration away from static payload types.

If the callee wants to neither send nor receive a stream offered by the caller, the callee sets the port number of that stream to zero in its media description.

There currently is no other way than port zero for the callee to refuse a bidirectional stream offered by the caller. Both caller and callee need to be aware what media tools are to be started.

For example, assume that the caller Alice has included the following description in her INVITE request. It includes an audio stream and two bidirectional video streams, using H.261 (payload type 31) and MPEG (payload type 32).

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.anywhere.com
c=IN IP4 host.anywhere.com
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 51372 RTP/AVP 31
a=rtpmap:31 H261/90000
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

The callee, Bob, does not want to receive or send the first video stream, so it returns the media description below:

```
v=0
o=bob 2890844730 2890844730 IN IP4 host.example.com
c=IN IP4 host.example.com
m=audio 47920 RTP/AVP 0 1
a=rtpmap:0 PCMU/8000
a=rtpmap:1 1016/8000
m=video 0 RTP/AVP 31
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

B.2 Setting SDP Values for Unicast

If a session description from a caller contains a media stream which is listed as send (receive) only, it means that the caller is only willing to send (receive) this stream, not receive (send). The same is true for the callee.

For receive-only and send-or-receive streams, the port number and address in the session description indicate where the media stream should be sent to by the recipient of the session description, either caller or callee. For send-only streams, the address and port number have no significance and SHOULD be set to zero.

The list of payload types for each media stream conveys two pieces of information, namely the set of codecs that the caller or callee is capable of sending or receiving, and the RTP payload type numbers used to identify those codecs. For receive-only or send-and-receive media streams, a caller SHOULD list all of the codecs it is capable of supporting in the session description in an INVITE or ACK. For send-only streams, the caller SHOULD indicate only those it wishes to send for this session. For receive-only streams, the payload type numbers indicate the value of the payload type field in RTP packets the caller is expecting to receive for that codec type. For send-only streams, the payload type numbers indicate the value of the payload type field in RTP packets the caller is planning to send for that codec type. For send-and-receive streams, the payload type numbers indicate the value of the payload type field the caller expects to both send and receive.

If a media stream is listed as receive-only by the caller, the callee lists, in the response, those codecs it intends to use from among the ones listed in the request. If a media stream is listed as send-only by the caller, the callee lists, in the response, those codecs it is willing to receive among the ones listed in the the request. If the media stream is listed as both send and receive, the callee lists those codecs it is capable of sending or receiving among the ones listed by the caller in the INVITE. The actual payload type numbers in the callee's session description corresponding to a particular codec MUST be the same as the caller's session description.

If caller and callee have no media formats in common for a particular stream, the callee MUST return a session description containing the particular "m=" line, but with the port number set to zero, and no payload types listed.

If there are no media formats in common for all streams, the callee SHOULD return a 400 response, with a 304 Warning header field.

B.3 Multicast Operation

The interpretation of send-only and receive-only for multicast media sessions differs from that for unicast sessions. For multicast, send-only means that the recipient of the session description (caller or callee) SHOULD only send media streams to the address and port indicated. Receive-only means that the recipient of the session description SHOULD only receive media on the address and port indicated.

For multicast, receive and send multicast addresses are the same and all parties use the same port numbers to receive media data. If the session description provided by the caller is acceptable to the callee, the callee can choose not to include a session description or MAY echo the description in the response.

A callee MAY, in the response, return a session description with some of the payload types removed, or port numbers set to zero (but no other value). This indicates to the caller that the callee does not support the given stream or media types which were removed. A callee MUST NOT change whether a given stream is send-only, receive-only, or send-and-receive.

If a callee does not support multicast at all, it SHOULD return a 400 status response and include a 330 Warning.

B.4 Delayed Media Streams

In some cases, a caller may not know the set of media formats which it can support at the time it would like to issue an invitation. This is the case when the caller is actually a gateway to another protocol which performs media format negotiation after call setup. When this occurs, a caller MAY issue an INVITE with a session description that contains no media lines. The callee SHOULD interpret this to mean that the caller wishes to participate in a multimedia session described by the session description, but that the media streams are not yet known. The callee SHOULD return a session description indicating the streams and media formats it is willing to support, however. The caller MAY update the session description either in the ACK request or in a re-INVITE at a later time, once the streams are known.

B.5 Putting Media Streams on Hold

If a party in a call wants to put the other party "on hold", i.e., request that it temporarily stops sending one or more media streams, a party re-invites the other by sending an INVITE request with a modified session description. The session description is the same as

in the original invitation (or response), but the "c" destination addresses for the media streams to be put on hold are set to zero (0.0.0.0).

B.6 Subject and SDP "s=" Line

The SDP "s=" line and the SIP Subject header field have different meanings when inviting to a multicast session. The session description line describes the subject of the multicast session, while the SIP Subject header field describes the reason for the invitation. The example in Section 16.2 illustrates this point. For invitations to two-party sessions, the SDP "s=" line MAY be left empty.

B.7 The SDP "o=" Line

The "o=" line is not strictly necessary for two-party sessions, but MUST be present to allow re-use of SDP-based tools.

C Summary of Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [9]. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

```
name  =  definition
```

The name of a rule is simply the name itself (without any enclosing "<" and ">") and is separated from its definition by the equal "=" character. White space is only significant in that indentation of continuation lines is used to indicate a rule definition that spans more than one line. Certain basic rules are in uppercase, such as SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions whenever their presence will facilitate discerning the use of rule names.

"literal"

Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

rule1 | rule2

Elements separated by a bar ("|") are alternatives, e.g., "yes | no" will accept yes or no.

(rule1 rule2)

Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo | bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

*rule

The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two.

[rule]

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

N rule

Specific repetition: "<n>(element)" is equivalent to "<n>*<n>(element)"; that is, exactly <n> occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

#rule

A construct "#" is defined, similar to "*", for defining lists of elements. The full form is "<n>#<m> element" indicating at least <n> and at most <m> elements, each separated by one or more commas (",") and OPTIONAL linear white space (LWS). This makes the usual form of lists very easy; a rule such as

(*LWS element *(*LWS "," *LWS element))

can be shown as 1# element. Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element), , (element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element MUST be present. Default values are 0 and infinity so that "#element" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

```
; comment
```

A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

```
implied *LWS
```

The grammar described by this specification is word-based. Except where noted otherwise, linear white space (LWS) can be included between any two adjacent words (token or quoted-string), and between adjacent tokens and separators, without changing the interpretation of a field. At least one delimiter (LWS and/or separators) MUST exist between any two tokens (for the definition of "token" below), since they would otherwise be interpreted as a single token.

C.1 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986.

```
OCTET     =  <any 8-bit sequence of data>
CHAR      =  <any US-ASCII character (octets 0 - 127)>
upalpha   =  "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
             "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
             "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
lowalpha  =  "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
             "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
             "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
alpha     =  lowalpha | upalpha
digit     =  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
             "8" | "9"
alphanum  =  alpha | digit
CTL       =  <any US-ASCII control character
             (octets 0 -- 31) and DEL (127)>
CR        =  %d13 ; US-ASCII CR, carriage return character
LF        =  %d10 ; US-ASCII LF, line feed character
SP        =  %d32 ; US-ASCII SP, space character
HT        =  %d09 ; US-ASCII HT, horizontal tab character
CRLF      =  CR LF ; typically the end of a line
```

The following are defined in RFC 2396 [12] for the SIP URI:

```
unreserved  =  alphanum | mark
mark        =  "-" | "_" | "." | "!" | "~" | "*" | "'"
            |  "(" | ")"
escaped     =  "%" hex hex
```

SIP header field values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream.

```
LWS  =  [CRLF] 1*( SP | HT ) ; linear whitespace
```

The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 character set (RFC 2279 [21]). In this regard, SIP differs from HTTP, which uses the ISO 8859-1 character set.

```
TEXT-UTF8  =  <any UTF-8 character encoding, except CTLs,
              but including LWS>
```

A CRLF is allowed in the definition of TEXT-UTF8 only as part of a header field continuation. It is expected that the folding LWS will be replaced with a single SP before interpretation of the TEXT-UTF8 value.

Hexadecimal numeric characters are used in several protocol elements.

```
hex  =  "A" | "B" | "C" | "D" | "E" | "F"
        | "a" | "b" | "c" | "d" | "e" | "f" | digit
```

Many SIP header field values consist of words separated by LWS or special characters. These special characters MUST be in a quoted string to be used within a parameter value.

```
token       =  1*< any CHAR  except CTL's  or separators>
separators  =  "(" | ")" | "<" | ">" | "@" |
               "," | ";" | ":" | "\" | <"> |
               "/" | "[" | "]" | "?" | "=" |
               "{" | "}" | SP | HT
```

Comments can be included in some SIP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition. In all other fields, parentheses are considered part of the field value.

```
comment  =  "(" *(ctext | quoted-pair | comment) ")"
ctext    =  < any TEXT-UTF8  excluding "("  and ")">
```

A string of text is parsed as a single word if it is quoted using double-quote marks.

```
quoted-string  =  ( <"> *(qdtext | quoted-pair ) <"> )
qdtext         =  <any TEXT-UTF8 except <">>
```

The backslash character ("\") MAY be used as a single-character quoting mechanism only within quoted-string and comment constructs.

```
quoted-pair  =  " \ " CHAR
```

D Using SRV DNS Records

The following procedure is experimental and relies on DNS SRV records (RFC 2052 [14]). The steps listed below are used in place of the two steps in section 1.4.2.

If a step elicits no addresses, the client continues to the next step. However if a step elicits one or more addresses, but no SIP server at any of those addresses responds, then the client concludes the server is down and doesn't continue on to the next step.

When SRV records are to be used, the protocol to use when querying for the SRV record is "sip". SRV records contain port numbers for servers, in addition to IP addresses; the client always uses this port number when contacting the SIP server. Otherwise, the port number in the SIP URI is used, if present. If there is no port number in the URI, the default port, 5060, is used.

1. If the host portion of the Request-URI is an IP address, the client contacts the server at the given address. If the host portion of the Request-URI is not an IP address, the client proceeds to the next step.

2. The Request-URI is examined. If it contains an explicit port number, the next two steps are skipped.

3. The Request-URI is examined. If it does not specify a protocol (TCP or UDP), the client queries the name server for SRV records for both UDP (if supported by the client) and TCP (if supported by the client) SIP servers. The format of these queries is defined in RFC 2052 [14]. The results of the query or queries are merged together and ordered based on priority. Then, the searching technique outlined in RFC 2052 [14] is used to select servers in order. If DNS doesn't return any records, the user goes to the last step. Otherwise, the user attempts to contact each server in the order listed. If no server is contacted, the user gives up.

4. If the Request-URI specifies a protocol (TCP or UDP) that is supported by the client, the client queries the name server for SRV records for SIP servers of that protocol type only. If the client does not support the protocol specified in the Request-URI, it gives up. The searching technique outlined in RFC 2052 [14] is used to select servers from the DNS response in order. If DNS doesn't

return any records, the user goes to the last step. Otherwise, the user attempts to contact each server in the order listed. If no server is contacted, the user gives up.

5. The client queries the name server for address records for the host portion of the Request-URI. If there were no address records, the client stops, as it has been unable to locate a server. By address record, we mean A RR's, AAAA RR's, or their most modern equivalent.

A client MAY cache a successful DNS query result. A successful query is one which contained records in the answer, and a server was contacted at one of the addresses from the answer. When the client wishes to send a request to the same host, it starts the search as if it had just received this answer from the name server. The server uses the procedures specified in RFC1035 [15] regarding cache invalidation when the time-to-live of the DNS result expires. If the client does not find a SIP server among the addresses listed in the cached answer, it starts the search at the beginning of the sequence described above.

For example, consider a client that wishes to send a SIP request. The Request-URI for the destination is sip:user@company.com. The client only supports UDP. It would follow these steps:

1. The host portion is not an IP address, so the client goes to step 2 above.

2. The client does a DNS query of QNAME="sip.udp.company.com", QCLASS=IN, QTYPE=SRV. Since it doesn't support TCP, it omits the TCP query. There were no addresses in the DNS response, so the client goes to the next step.

3. The client does a DNS query for A records for "company.com". An address is found, so that client attempts to contact a server at that address at port 5060.

E IANA Considerations

Section 4.4 describes a name space and mechanism for registering SIP options.

Section 6.41 describes the name space for registering SIP warn-codes.

F Acknowledgments

We wish to thank the members of the IETF MMUSIC WG for their comments and suggestions. Detailed comments were provided by Anders Kristensen, Jim Buller, Dave Devanathan, Yaron Goland, Christian Huitema, Gadi Karmi, Jonathan Lennox, Keith Moore, Vern Paxson, Moshe J. Sambol, and Eric Tremblay.

This work is based, inter alia, on [37,38].

G Authors' Addresses

Mark Handley
AT&T Center for Internet Research at ISCI (ACIRI)
1947 Center St., Suite 600
Berkeley, CA 94704-119
USA
Email: mjh@aciri.org

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
Email: schulzrinne@cs.columbia.edu

Eve Schooler
Computer Science Department 256-80
California Institute of Technology
Pasadena, CA 91125
USA
Email: schooler@cs.caltech.edu

Jonathan Rosenberg
Lucent Technologies, Bell Laboratories
Rm. 4C-526
101 Crawfords Corner Road
Holmdel, NJ 07733
USA
Email: jdrosen@bell-labs.com

H Bibliography

[1] Pandya, R., "Emerging mobile and personal communication systems," IEEE Communications Magazine , vol. 33, pp. 44--52, June 1995.

[2] Braden, B., Zhang, L., Berson, S., Herzog, S. and S. Jamin, "Resource ReSerVation protocol (RSVP) -- version 1 functional specification", RFC 2205, October 1997.

[3] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: a transport protocol for real-time applications", RFC 1889, Internet Engineering Task Force, Jan. 1996.

[4] Schulzrinne, H., Lanphier, R. and A. Rao, "Real time streaming protocol (RTSP)", RFC 2326, April 1998.

[5] Handley, M., "SAP: Session announcement protocol," Internet Draft, Internet Engineering Task Force, Nov. 1996. Work in progress.

[6] Handley, M. and V. Jacobson, "SDP: session description protocol", RFC 2327, April 1998.

[7] International Telecommunication Union, "Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.

[8] International Telecommunication Union, "Control protocol for multimedia communication," Recommendation H.245, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[9] International Telecommunication Union, "Media stream packetization and synchronization on non-guaranteed quality of service LANs," Recommendation H.225.0, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1996.

[10] Bradner, S., "Key words for use in RFCs to indicate requirement levels", BCP 14, RFC 2119, Mardch 1997.

[11] Fielding, R., Gettys, J., Mogul, J., Nielsen, H. and T. Berners-Lee, "Hypertext transfer protocol -- HTTP/1.1", RFC 2068, January 1997.

[12] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform resource identifiers (URI): generic syntax", RFC 2396, August 1998.

[13] Berners-Lee, T., Masinter, L. and M. McCahill, "Uniform resource locators (URL)", RFC 1738, December 1994.

[14] Gulbrandsen, A. and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996.

[15] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, Noveberm 1997.

[16] Hamilton, M. and R. Wright, "Use of DNS aliases for network services", RFC 2219, October 1997.

[17] Zimmerman, D., "The finger user information protocol", RFC 1288, December 1991.

[18] Williamson, S., Kosters, M., Blacka, D., Singh, J. and K. Zeilstra, "Referral whois (rwhois) protocol V1.5", RFC 2167, June 1997.

[19] Yeong, W., Howes, T. and S. Kille, "Lightweight directory access protocol", RFC 1777, March 1995.

[20] Schooler, E., "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.

[21] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[22] Stevens, W., TCP/IP illustrated: the protocols , vol. 1. Reading, Massachusetts: Addison-Wesley, 1994.

[23] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[24] Crocker, D., "Standard for the format of ARPA internet text messages", RFC STD 11, RFC 822, August 1982.

[25] Meyer, D., "Administratively scoped IP multicast", RFC 2365, July 1998.

[26] Schulzrinne, H., "RTP profile for audio and video conferences with minimal control", RFC 1890, January 1996

[27] Eastlake, D., Crocker, S. and J. Schiller, "Randomness recommendations for security", RFC 1750, December 1994.

[28] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL scheme", RFC 2368, July 1998.

[29] Braden, B., "Requirements for internet hosts - application and support", STD 3, RFC 1123, October 1989.

[30] Palme, J., "Common internet message headers", RFC 2076, February 1997.

[31] Alvestrand, H., "IETF policy on character sets and languages", RFC 2277, January 1998.

[32] Elkins, M., "MIME security with pretty good privacy (PGP)", RFC 2015, October 1996.

[33] Atkins, D., Stallings, W. and P. Zimmermann, "PGP message exchange formats", RFC 1991, August 1996.

[34] Atkinson, R., "Security architecture for the internet protocol", RFC 2401, November 1998.

[35] Allen, C. and T. Dierks, "The TLS protocol version 1.0," RFC 2246, January 1999.

[36] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP authentication: Basic and digest access authentication," Internet Draft, Internet Engineering Task Force, Sept. 1998. Work in progress.

[37] Schooler, E., "Case study: multimedia conference control in a packet-switched teleconferencing system," Journal of Internetworking: Research and Experience , vol. 4, pp. 99--120, June 1993. ISI reprint series ISI/RS-93-359.

[38] Schulzrinne, H., "Personal mobility for multimedia services in the Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS) , (Berlin, Germany), Mar. 1996.

Full Copyright Statement

Superseded by a more recent version

INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.323

(11/96)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Systems and terminal equipment for audiovisual services

---

Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service

ITU-T Recommendation H.323

Superseded by a more recent version

(Previously CCITT Recommendation)

Superseded by a more recent version

ITU-T H-SERIES RECOMMENDATIONS

AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Characteristics of transmission channels used for other than telephone purposes	H.10–H.19
Use of telephone-type circuits for voice-frequency telegraphy	H.20–H.29
Telephone circuits or cables used for various types of telegraph transmission or simultaneous transmission	H.30–H.39
Telephone-type circuits used for facsimile telegraphy	H.40–H.49
Characteristics of data signals	H.50–H.99
CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100–H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	H.200–H.399
General	H.200–H.219
Transmission multiplexing and synchronization	H.220–H.229
Systems aspects	H.230–H.239
Communication procedures	H.240–H.259
Coding of moving video	H.260–H.279
Related systems aspects	H.280–H.299
Systems and terminal equipment for audiovisual services	**H.300–H.399**

For further details, please refer to ITU-T List of Recommendations.

Superseded by a more recent version

ITU-T RECOMMENDATION H.323

VISUAL TELEPHONE SYSTEMS AND EQUIPMENT FOR LOCAL AREA NETWORKS WHICH PROVIDE A NON-GUARANTEED QUALITY OF SERVICE

Summary

Recommendation H.323 describes terminals, equipment and services for multimedia communication over Local Area Networks (LAN) which do not provide a guaranteed quality of service. H.323 terminals and equipment may carry real-time voice, data and video, or any combination, including videotelephony.

The LAN over which H.323 terminals communicate, may be a single segment or ring, or it may be multiple segments with complex topologies. It should be noted that operation of H.323 terminals over the multiple LAN segments (including the Internet) may result in poor performance. The possible means by which quality of service might be assured on such types of LANs/internetworks is beyond the scope of this Recommendation.

H.323 terminals may be integrated into personal computers or implemented in stand-alone devices such as videotelephones. Support for voice is mandatory, while data and video are optional, but if supported, the ability to use a specified common mode of operation is required, so that all terminals supporting that media type can interwork. This Recommendation allows more than one channel of each type to be in use. Other Recommendations in the H.323-Series include H.225.0 packet and synchronization, H.245 control, H.261 and H.263 video codecs, G.711, G.722, G.728, G.729, and G.723 audio codecs, and the T.120-Series of multimedia communications protocols.

This Recommendation makes use of the logical channel signalling procedures of Recommendation H.245, in which the content of each logical channel is described when the channel is opened. Procedures are provided for expression of receiver and transmitter capabilities, so transmissions are limited to what receivers can decode, and so that receivers may request a particular desired mode from transmitters. Since the procedures of Recommendation H.245 are also used by Recommendation H.310 for ATM networks, Recommendation H.324 for GSTN, and V.70, interworking with these systems should not require H.242 to H.245 translation as would be the case for H.320 systems.

H.323 terminals may be used in multipoint configurations, and may interwork with H.310 terminals on B-ISDN, H.320 terminals on N-ISDN, H.321 terminals on B-ISDN, H.322 terminals on Guaranteed Quality of Service LANs, H.324 terminals on GSTN and wireless networks, and V.70 terminals on GSTN.

Source

ITU-T Recommendation H.323 was prepared by ITU-T Study Group 15 (1993-1996) and was approved under the WTSC Resolution No. 1 procedure on the 8th of November 1996.

Superseded by a more recent version

FOREWORD

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the ITU. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, the ITU had/had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

Superseded by a more recent version

CONTENTS

Superseded by a more recent version

Recommendation H.323

VISUAL TELEPHONE SYSTEMS AND EQUIPMENT FOR LOCAL AREA NETWORKS WHICH PROVIDE A NON-GUARANTEED QUALITY OF SERVICE

(Geneva, 1996)

1 Scope

This Recommendation covers the technical requirements for narrow-band visual telephone services defined in H.200/AV.120-Series Recommendations, in those situations where the transmission path includes one or more Local Area Networks (LAN), which may not provide a guaranteed Quality of Service (QOS) equivalent to that of N-ISDN. Examples of this type of LAN are:

– Ethernet (IEEE 802.3);

– Fast Ethernet (IEEE 802.10);

– FDDI (non-guaranteed quality of service mode);

– Token Ring (IEEE 802.5).

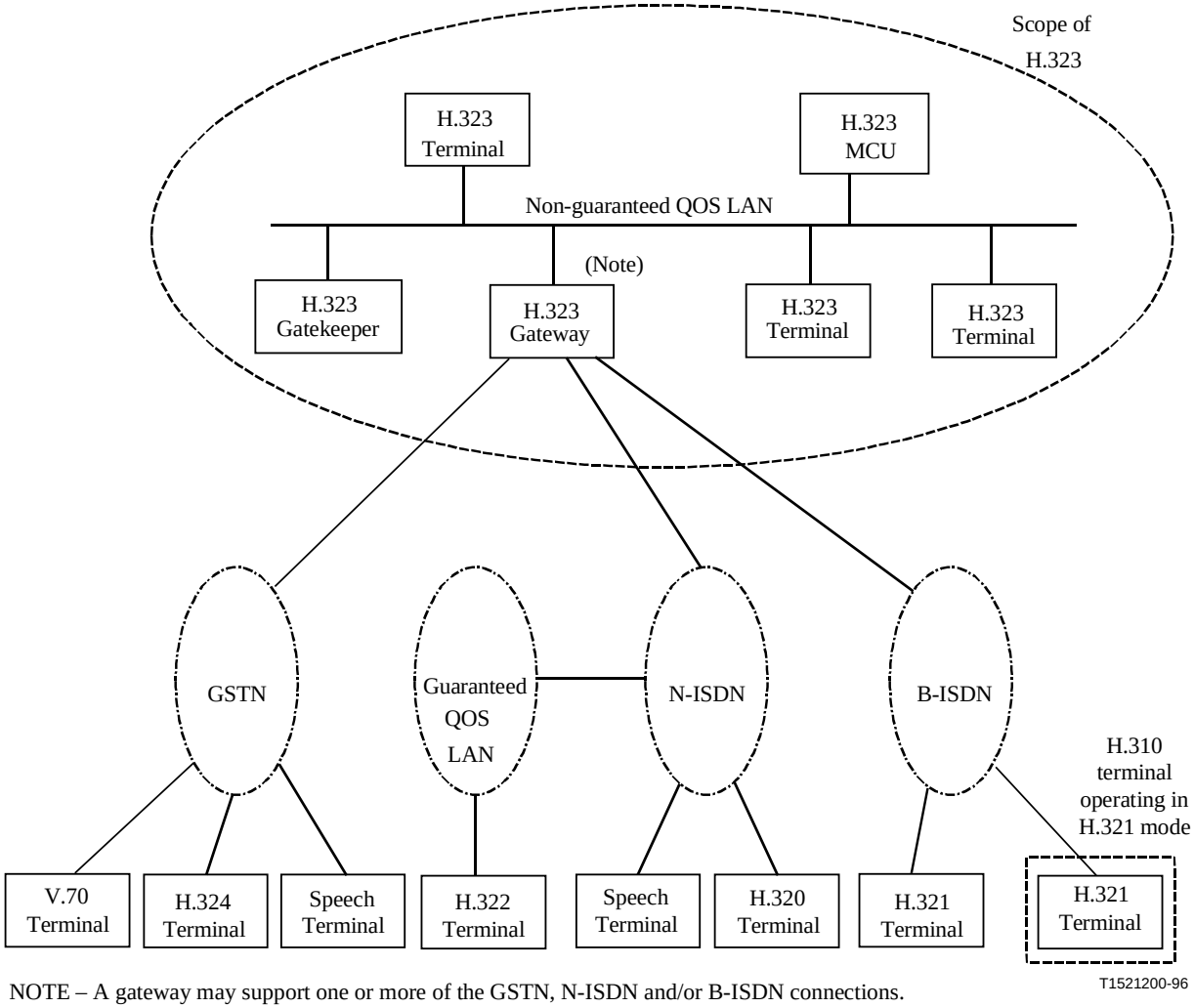
This Recommendation covers the case of visual telephone services in those situations where the transmission path includes one or more Local Area Networks (LAN), which are configured and managed to provide a guaranteed Quality of Service (QOS) equivalent to that of N-ISDN such that no additional protection or recovery mechanisms beyond those mandated by Recommendation H.320 need to be provided in the terminals. Pertinent parameters are the data error and loss properties and variation of transit delay. An example of a suitable LAN is: Integrated Services (IS) LAN: IEEE 802.9A Isochronous services with Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Media Access Control (MAC) service.

H.323 terminals may be used in multipoint configurations, and may interwork with H.310 terminals on B-ISDN, H.320 terminals on N-ISDN, H.321 terminals on B-ISDN, H.322 terminals on Guaranteed Quality of Service LANs, H.324 terminals on GSTN and wireless networks, and V.70 terminals on GSTN. See Figure 1.

The scope of this Recommendation does not include the LAN itself, or the transport layer which may be used to connect various LANs. Only elements needed for interaction with the Switched Circuit Network (SCN) are within the scope of this Recommendation. The combination of the H.323 Gateway, the H.323 terminal, and the out-of-scope LAN appears on the SCN as an H.320, H.310, H.324, or V.70 terminal.

This Recommendation describes the components of an H.323 system. This includes Terminals, Gateways, Gatekeepers, Multipoint Controllers, Multipoint Processors and Multipoint Control Units. Control messages and procedures within this Recommendation define how these components communicate. Detailed descriptions of these components are contained in clause 6.

The components described in this Recommendation consist of H.323 endpoints and H.323 entities. The endpoints can call and are callable according to the call set-up procedures of clause 8. The entities are not callable however, they can be addressed in order to perform their specific functions. For example, a terminal cannot place a call to a Gatekeeper however, the Gatekeeper is addressed as part of the call establishment procedures.



NOTE – A gateway may support one or more of the GSTN, N-ISDN and/or B-ISDN connections.

Figure 1/H.323 – Interoperability of H.323 terminals

2 Normative references

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

[1] ITU-T Recommendation H.225.0 (1996), *Media stream packetization and synchronization for visual telephone systems on non-guaranteed quality of service LANs*.

[2] ITU-T Recommendation H.245 (1996), *Control protocol for multimedia communications*.

[3] CCITT Recommendation G.711 (1988), *Pulse Code Modulation (PCM) of voice frequencies*.

[4] CCITT Recommendation G.722 (1988), *7 kHz audio-coding within 64 kbit/s*.

[5] ITU-T Recommendation G.723.1 (1996), *Speech coders: Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s*.

[6] CCITT Recommendation G.728 (1992), *Coding of speech at 16 kbit/s using low-delay code excited linear prediction*.

[7] ITU-T Recommendation G.729 (1996), *Coding of speech at 8 kbit/s using conjugate structure algebraic-code-excited linear-prediction (CS-ACELP)*.

[8] ITU-T Recommendation H.261 (1993*), Video codec for audiovisual services at p* × *64 kbit/s*.

[9] ITU-T Recommendation H.263 (1996), *Video coding for low bit rate communication*.

[10] ITU-T Recommendation T.120 (1996), *Transmission protocols for multimedia data*.

[11] ITU-T Recommendation H.320 (1996), *Narrow-band visual telephone systems and terminal equipment*.

[12] ITU-T Recommendation H.321 (1996), *Adaptation of H.320 visual telephone terminals to B-ISDN environments*.

[13] ITU-T Recommendation H.322 (1996), *Visual telephone systems and terminal equipment for local area networks which provide a guaranteed quality of service*.

[14] ITU-T Recommendation H.324 (1996), *Terminal for low bit rate multimedia communication*.

[15] ITU-T Recommendation H.310 (1996), *Broadband and audiovisual communication systems and terminals*.

[16] ITU-T Recommendation Q.931 (1993), *ISDN user-network interface layer 3 specification for basic call control*.

[17] ITU-T Recommendation Q.932 (1993), *Generic procedures for the control of ISDN supplementary services*.

[18] ITU-T Recommendation Q.950 (1993), *Supplementary services protocols, structure and general principles*.

[19] ISO/IEC 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (USC) – Part I: Architecture and Basic Multilingual Plane*.

[20] CCITT Recommendation E.164 (1991), *Numbering plan for the ISDN era*.

3 Definitions

For the purposes of this Recommendation the definitions given in clause 3/H.225.0 [1] and clause 3/H.245 [2] apply along with those in this clause. These definitions apply to the LAN side only. Other terms may be appropriate when referring to the Switched Circuit Network (SCN) side.

3.1 active MC: An MC that has won the master-slave determination procedure and is currently providing the multipoint control function for the conference.

3.2 ad hoc multipoint conference: An Ad Hoc Multipoint conference was a point-to-point conference that had been expanded into a multipoint conference at some time during the call. This can be done if one or more of the terminals in the initial point-to-point conference contains an MC, if the call is made using a Gatekeeper that includes MC functionality, or if the initial call is made through an MCU as a multipoint call between only two terminals.

3.3 addressable: An H.323 entity on the LAN having a Transport Address. Not the same as being callable. A terminal or MCU is addressable and callable. A Gatekeeper is addressable but not callable. An MC or MP is neither callable nor addressable but is contained within an endpoint or Gatekeeper that is.

3.4 audio mute: Suppressing of the audio signal of a single or all source(s). Send muting means that the originator of an audio stream mutes its microphone and/or does not transmit any audio signal

at all. Receive mute means that the receiving terminal ignores a particular incoming audio stream or mutes its loudspeaker.

3.5 broadcast conference: A Broadcast conference is one in which there is one transmitter of media streams and many receivers. There is no bidirectional transmission of control or media streams. Such conferences should be implemented using LAN transport multicast facilities, if available. This conference type is for further study.

3.6 broadcast panel conference: A Broadcast Panel conference is a combination of a Multipoint conference and a Broadcast conference. In this conference, several terminals are engaged in a multipoint conference, while many other terminals are only receiving the media streams. There is bidirectional transmission between the terminals in the multipoint portion of the conference and no bidirectional transmission between them and the listening terminals. This conference type is for further study.

3.7 call: Point-to-point multimedia communication between two H.323 endpoints. The call begins with the call set-up procedure and ends with the call termination procedure. The call consists of the collection of reliable and unreliable channels between the endpoints. In case of interworking with some SCN endpoints via a gateway, all the channels terminate at the Gateway where they are converted to the appropriate representation for the SCN end system.

3.8 call signalling channel: Reliable channel used to convey the call set-up and teardown messages (following Recommendation H.225.0) between two H.323 entities.

3.9 callable: Capable of being called as described in clause 8. Terminals, MCUs and Gateways are callable, but Gatekeepers and MCs are not.

3.10 centralized multipoint conference: A Centralized Multipoint conference is one in which all participating terminals communicate in a point-to-point fashion with an MCU. The terminals transmit their control, audio, video, and/or data streams to the MCU. The MC within the MCU centrally manages the conference. The MP within the MCU processes the audio, video, and/or data streams, and returns the processed streams to each terminal.

3.11 control and indication (C&I): End-to-end signalling between terminals, consisting of Control, which causes a state change in the receiver, and Indication which provides for information as to the state or functioning of the system (see also Recommendation H.245 [2] for additional information and abbreviations).

3.12 data: Information stream other than audio, video, and control, carried in the logical data channel (see Recommendation H.225.0 [1]).

3.13 decentralized multipoint conference: A Decentralized Multipoint conference is one in which the participating terminals multicast their audio and video to all other participating terminals without using an MCU. The terminals are responsible for:

a) summing the received audio streams; and

b) selecting one or more of the received video streams for display.

No audio or video MP is required in this case. The terminals communicate on their H.245 Control Channels with an MC which manages the conference. The data stream is still centrally processed by the MCS-MCU which may be within an MP.

3.14 endpoint: An H.323 terminal, Gateway, or MCU. An endpoint can call and be called. It generates and/or terminates information streams.

3.15 gatekeeper: The Gatekeeper (GK) is an H.323 entity on the LAN that provides address translation and controls access to the local area network for H.323 terminals, Gateways and MCUs.

The Gatekeeper may also provide other services to the terminals, Gateways and MCUs such as bandwidth management and locating Gateways.

3.16 gateway: An H.323 Gateway (GW) is an endpoint on the local area network which provides for real-time, two-way communications between H.323 Terminals on the LAN and other ITU Terminals on a wide area network, or to another H.323 Gateway. Other ITU Terminals include those complying with Recommendations H.310 (H.320 on B-ISDN), H.320 (ISDN), H.321 (ATM), H.322 (GQOS-LAN), H.324 (GSTN), H.324M (Mobile), and V.70 (DSVD).

3.17 H.323 Entity: Any H.323 component, including terminals, Gateways, Gatekeepers, MCs, MPs, and MCUs.

3.18 H.245 control channel: Reliable Channel used to carry the H.245 control information messages (following H.245) between two H.323 endpoints.

3.19 H.245 logical channel: Channel used to carry the information streams between two H.323 endpoints. These channels are established following the H.245 OpenLogicalChannel procedures. An unreliable channel is used for audio, audio control, video, and video control information streams. A reliable channel is used for data and H.245 control information streams. There is no relationship between a logical channel and a physical channel.

3.20 H.245 session: The part of the call that begins with the establishment of an H.245 Control Channel, and ends with the receipt of the H.245 **EndSessionCommand** or termination due to failure. Not to be confused with a call, which is delineated by the H.225.0 Setup and Release Complete messages.

3.21 hybrid multipoint conference – centralized audio: A Hybrid Multipoint – Centralized Audio conference is one in which terminals multicast their video to other participating terminals, and unicast their audio to the MP for mixing. The MP returns a mixed audio stream to each terminal.

3.22 hybrid multipoint conference – centralized video: A Hybrid Multipoint – Centralized Video conference is one in which terminals multicast their audio to other participating terminals, and unicast their video to the MP for switching or mixing. The MP returns a video stream to each terminal.

3.23 information stream: A flow of information of a specific media type (e.g. audio) from a single source to one or more destinations.

3.24 LAN address: The network layer address of an H.323 entity as defined by the (inter)network layer protocol in use (e.g. an IP address). This address is mapped onto the layer one address of the respective system by some means defined in the (inter)networking protocol.

3.25 lip synchronization: Operation to provide the feeling that speaking motion of the displayed person is synchronized with his speech.

3.26 local area network (LAN): A shared or switched medium, peer-to-peer communications network that broadcasts information for all stations to receive within a moderate-sized geographic area, such as a single office building or a campus. The network is generally owned, used, and operated by a single organization. In the context of Recommendation H.323, LANs also include internetworks composed of several LANs that are interconnected by bridges or routes.

3.27 mixed multipoint conference: A Mixed Multipoint conference (see Figure 2) has some terminals (D, E and F) participating in a centralized mode while other terminals (A, B and C) are participating in a decentralized mode. A terminal is not aware of the mixed nature of the conference, only of the type of conference it is participating in. The MCU provides the bridge between the two types of conferences.

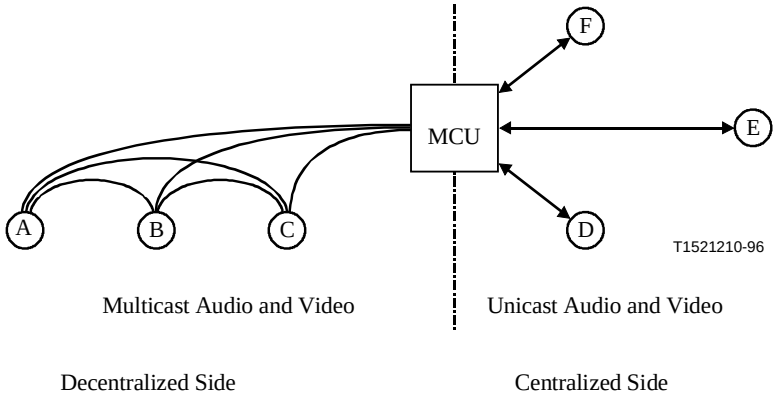


Figure 2/H.233 – Mixed multipoint conference

3.28 multicast: A process of transmitting PDUs from one source to many destinations. The actual mechanism (i.e. IP multicast, multi-unicast, etc.) for this process may be different for different LAN technologies.

3.29 multipoint conference: A Multipoint conference is a conference between three or more terminals. The terminals may be on the LAN or on the SCN. The multipoint conference shall always be controlled by an MC. Various multipoint conference types are defined in this clause but they all require a single MC per conference. They may also involve one or more H.231 MCUs on the SCN. A terminal on the LAN may also participate in an SCN multipoint conference by connecting via a Gateway to an SCN MCU. This does not require the use of an MC.

3.30 multipoint control unit: The Multipoint Control Unit (MCU) is an endpoint on the local area network which provides the capability for three or more terminals and Gateways to participate in a multipoint conference. It may also connect two terminals in a point-to-point conference which may later develop into a multipoint conference. The MCU generally operates in the fashion of an H.231 MCU, however, an audio processor is not mandatory. The MCU consists of two parts: a mandatory Multipoint Controller and optional Multipoint Processors. In the simplest case, an MCU may consist only of an MC with no MPs.

3.31 multipoint controller: The Multipoint Controller (MC) is an H.323 entity on the local area network which provides for the control of three or more terminals participating in a multipoint conference. May also connect two terminals in a point-to-point conference which may later develop into a multipoint conference. The MC provides for capability negotiation with all terminals to achieve common levels of communications. It also may control conference resources such as who is multicasting video. The MC does not perform mixing or switching of audio, video and data.

3.32 multipoint processor: The Multipoint Processor (MP) is an H.323 entity on the local area network which provides for the centralized processing of audio, video, and/or data streams in a multipoint conference. The MP provides for the mixing, switching, or other processing of media streams under the control of the MC. The MP may process a single media stream or multiple media streams depending on the type of conference supported.

3.33 multi-unicast: A process of transferring PDUs where an endpoint sends more than one copy of a media stream, but to different endpoints. This may be necessary in networks which do not support multicast.

3.34 point-to-point conference: A Point-to-Point conference is a conference between two terminals. It may be either directly between two H.323 terminals or between an H.323 terminal and an SCN terminal via a gateway. A call between two terminals (see Call).

3.35 RAS channel: Unreliable channel used to convey the registration, admissions, bandwidth change, and status messages (following Recommendation H.225.0) between two H.323 entities.

3.36 reliable channel: A transport connection used for reliable transmission of an information stream from its source to one or more destinations.

3.37 reliable transmission: Transmission of messages from a sender to a receiver using connection-mode data transmission. The transmission service guarantees sequenced, error-free, flow-controlled transmission of messages to the receiver for the duration of the transport connection.

3.38 RTP session: For each participant, the session is defined by a particular pair of destination Transport Addresses (one LAN address plus a TSAP identifier pair for RTP and RTCP). The destination Transport Address pair may be common for all participants, as in the case of IP multicast, or may be different for each, as in the case of individual unicast network addresses. In a multimedia session, the media audio and video are carried in separate RTP sessions with their own RTCP packets. The multiple RTP sessions are distinguished by different transport addresses.

3.39 switched circuit network (SCN): A public or private switched telecommunications network such as the GSTN, N-ISDN, or B-ISDN.

3.40 terminal: An H.323 Terminal is an endpoint on the local area network which provides for real-time, two-way communications with another H.323 terminal, Gateway, or Multipoint Control Unit. This communication consists of control, indications, audio, moving colour video pictures, and/or data between the two terminals. A terminal may provide speech only, speech and data, speech and video, or speech, data and video.

3.41 transport address: The transport layer address of an addressable H.323 entity as defined by the (inter)network protocol suite in use. The Transport Address of an H.323 entity is composed of the LAN address plus the TSAP identifier of the addressable H.323 entity.

3.42 transport connection: An association established by a transport layer between two H.323 entities for the transfer of data. In the context of this Recommendation, a transport connection provides reliable transmission of information.

3.43 TSAP identifier: The piece of information used to multiplex several transport connections of the same type on a single H.323 entity with all transport connections sharing the same LAN address, (e.g. the port number in a TCP/UDP/IP environment). TSAP identifiers may be (pre)assigned statically by some international authority or may be allocated dynamically during the setup of a call. Dynamically assigned TSAP identifiers are of transient nature, i.e. their values are only valid for the duration of a single call.

3.44 unicast: A process of transmitting messages from one source to one destination.

3.45 unreliable channel: A logical communication path used for unreliable transmission of an information stream from its source to one or more destinations.

3.46 unreliable transmission: Transmission of messages from a sender to one or more receivers by means of connectionless-mode data transmission. The transmission service is *best-effort* delivery of the PDU, meaning that messages transmitted by the sender may be lost, duplicated, or received out of order by (any of) the receiver(s).

3.47 well-known TSAP identifier: A TSAP identifier that has been allocated by an (international) authority that is in charge of the assignment of TSAP identifiers for a particular (inter)networking protocol and the related transport protocols (e.g. the IANA for TCP and UDP port numbers). This identifier is guaranteed to be unique in the context of the respective protocol.

3.48 zone: A Zone (see Figure 3) is the collection of all terminals (Tx), Gateways (GW), and Multipoint Control Units (MCU) managed by a single Gatekeeper (GK). A Zone includes at least

one terminal, and may or may not include Gateways or MCUs. A Zone has one and only one Gatekeeper. A Zone may be independent of LAN topology and may be comprised of multiple LAN segments which are connected using routes (R) or other devices.

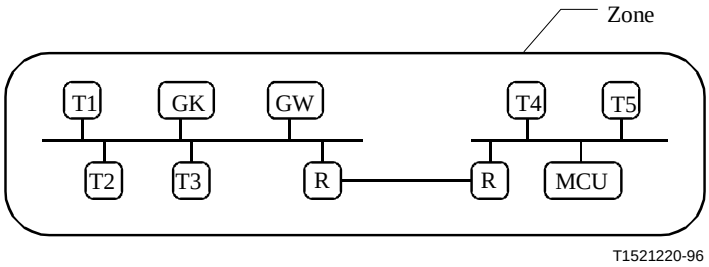


Figure 3/H.323 – Zone

4 Symbols and abbreviations

For the purpose of this Recommendation, the following symbols and abbreviations apply:

4CIF 4 times CIF

16CIF 16 times CIF

ACF Admission Confirmation

ARJ Admission Reject

ARQ Admission Request

BAS Bit rate Allocation Signal

BCF Bandwidth Change Confirmation

BCH Bose, Chaudhuri, and Hocquengham

B-ISDN Broadband-Integrated Services Digital Network

BRJ Bandwidth Change Reject

BRQ Bandwidth Change Request

C&I Control and Indication

CID Conference Identifier

CIF Common Intermediate Format

DCF Disengage Confirmation

DID Direct Inward Dialling

DRQ Disengage Request

ECS Encryption Control Signal

EIV Encryption Initialization Vector

FAS Frame Alignment Signal

FIR Full Intra Request

GCF Gatekeeper Confirmation

GK Gatekeeper

GQOS	Guaranteed Quality of Service
GRJ	Gatekeeper Reject
GRQ	Gatekeeper Request
GSTN	General Switched Telephone Network
GW	Gateway
IANA	Internet Assigned Number Authority
IP	Internet Protocol
IPX	Internetwork Protocol Exchange
IRQ	Information Request
IRR	Information Request Response
ISDN	Integrated Services Digital Network
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
LAN	Local Area Network
LCF	Location Confirmation
LCN	Logical Channel Number
LRJ	Location Reject
LRQ	Location Request
MC	Multipoint Controller
MCS	Multipoint Communications System
MCU	Multipoint Control Unit
MP	Multipoint Processor
MSN	Multiple Subscriber Number
N-ISDN	Narrow-band-Integrated Services Digital Network
NACK	Negative Acknowledge
QCIF	Quarter CIF
RCF	Registration Confirmation
RRJ	Registration Reject
RRQ	Registration Request
RTP	Real Time Protocol
RTCP	Real Time Control Protocol
SBE	Single Byte Extension
SCM	Selected Communications Mode
SCN	Switched Circuit Network
SPX	Sequential Protocol Exchange
SQCIF	Sub QCIF

TCP	Transport Control Protocol
TSAP	Transport Layer Service Access Point
UCF	Unregister Confirmation
UDP	User Datagram Protocol
URJ	Unregister Reject
URQ	Unregister Request

5 Conventions

In this Recommendation, the following conventions are used:

"Shall" indicates a mandatory requirement.

"Should" indicates a suggested but optional course of action.

"May" indicates an optional course of action rather than a recommendation that something take place.

References to clauses, subclauses, Annexes and Appendices refer to those items within this Recommendation unless another document is explicitly listed. For example, 1.4 refers to 1.4 of this Recommendation; 6.4/H.245 refers to 6.4 in Recommendation H.245.

Where items exist on both the LAN and the SCN, references to the SCN item will be explicit. For example, an MCU is an H.323 MCU on the LAN, an SCN-MCU is an MCU on the SCN.

This Recommendation describes the use of three different message types: H.245, RAS and Q.931. To distinguish between the different message types, the following conversion is followed. H.245 message and parameter names consist of multiple concatenated words highlighted in bold typeface (**maximumDelayJitter**). RAS message names are represented by three letter abbreviations (ARQ). Q.931 message names consist of one or two words with the first letters capitalized (Call Proceeding).

6 System description

This Recommendation describes the elements of the H.323 components. These are Terminals, Gateways, Gatekeepers, MCs and MCUs. These components communicate through the transmission of Information Streams. The characteristics of these components are described in this clause.

6.1 Information streams

Visual telephone components communicate through the transmission of Information Streams. These Information Streams are classified into video, audio, data, communications control and call control as follows.

Audio signals contain digitized and coded speech. In order to reduce the average bit rate of audio signals, voice activation may be provided. The audio signal is accompanied by an audio control signal.

Video signals contain digitized and coded motion video. Video is transmitted at a rate no greater than that selected as a result of the capability exchange. The video signal is accompanied by a video control signal.

Data signals include still pictures, facsimile, documents, computer files and other data streams.

Communications control signals pass control data between remote like functional elements and are used for capability exchange, opening and closing logical channels, mode control and other functions that are part of communications control.

Call control signals are used for call establishment, disconnect and other call control functions.

The information streams described above are formatted and sent to the network interface as described in Recommendation H.225.0.

6.2 Terminal characteristics

An example of an H.323 terminal is shown in Figure 4. The diagram shows the user equipment interfaces, video codec, audio codec, telematic equipment, H.225.0 layer, system control functions and the interface to the LAN. All H.323 terminals shall have a System Control Unit, H.225.0 layer, Network Interface and an Audio Codec Unit. The Video Codec Unit and User Data Applications are optional.

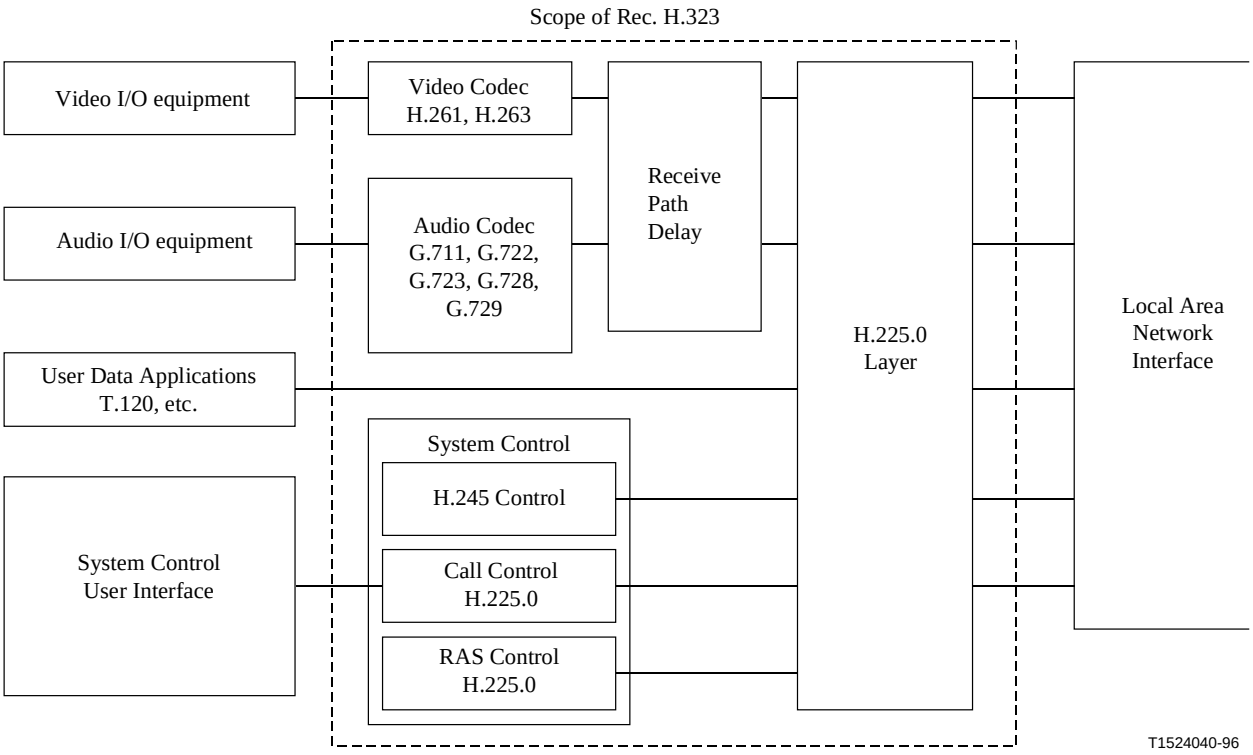


Figure 4/H.323 – H.323 terminal equipment

6.2.1 Terminal elements outside the scope of this Recommendation

The following elements are not within the scope of this Recommendation and are therefore not defined within this Recommendation:

- Attached audio devices providing voice activation sensing, microphone and loudspeaker, telephone instrument or equivalent, multiple microphones mixers, and acoustic echo cancellation.
- Attached video equipment providing cameras and monitors, and their control and selection, video processing to improve compression or provide split screen functions.
- Data applications and associated user interfaces which use T.120 or other data services over the data channel.
- Attached Network Interface, which provides the interface to the LAN, supporting appropriate signalling and voltage levels, in accordance with national and international standards.

- Human user system control, user interface and operation.

6.2.2 Terminal elements within the scope of this Recommendation

The following elements are within the scope of this Recommendation, and are therefore subject to standardization and are defined within this Recommendation:

- The Video Codec (H.261, etc.) encodes the video from the video source (i.e. camera) for transmission and decodes the received video code which is output to a video display.
- The Audio Codec (G.711, etc.) encodes the audio signal from the microphone for transmission and decodes the received audio code which is output to the loudspeaker.
- The Data Channel supports telematic applications such as electronic whiteboards, still image transfer, file exchange, database access, audiographics conferencing, etc. The standardized data application for real-time audiographics conferencing is Recommendation T.120. Other applications and protocols may also be used via H.245 negotiation as specified in 6.2.7.
- The System Control Unit (H.245, H.225.0) provides signalling for proper operation of the H.323 terminal. It provides for call control, capability exchange, signalling of commands and indications, and messages to open and fully describe the content of logical channels.
- H.225.0 Layer (H.225.0) formats the transmitted video, audio, data and control streams into messages for output to the network interface and retrieves the received video, audio, data, and control streams from messages which have been input from the network interface. In addition, it performs logical framing, sequence numbering, error detection and error correction as appropriate to each media type.

6.2.3 LAN interface

The LAN interface is implementation specific and is outside the scope of this Recommendation. However, the LAN interface shall provide the services described in Recommendation H.225.0. This includes the following: Reliable (e.g. TCP, SPX) end-to-end service is mandatory for the H.245 Control Channel, the Data Channels, and the Call Signalling Channel. Unreliable (e.g. UDP, IPX) end-to-end service is mandatory for the Audio Channels, the Video Channels, and the RAS Channel. These services may be duplex or simplex, unicast or multicast depending on the application, the capabilities of the terminals, and the configuration of the LAN.

6.2.4 Video codec

The video codec is optional. All H.323 terminals providing video communications shall be capable of encoding and decoding video according to H.261 QCIF. Optionally, a terminal may also be capable of encoding and decoding video according to H.261 CIF or H.263 SQCIF, QCIF, CIF, 4CIF, and 16CIF. If a terminal supports H.263 with CIF or higher resolution, it shall also support H.261 CIF. All terminals which support H.263 shall support H.263 QCIF. The H.261 and H.263 codecs, on the LAN, shall be used without BCH error correction and without error correction framing.

Other video codecs, and other picture formats, may also be used via H.245 negotiation. More than one video channel may be transmitted and/or received, as negotiated via the H.245 Control Channel. The H.323 terminal may optionally send more than one video channel at the same time, for example, to convey the speaker and a second video source. The H.323 terminal may optionally receive more than one video channel at the same time, for example, to display multiple participants in a distributed multipoint conference.

CIF and QCIF are defined in Recommendation H.261. SQCIF, 4CIF and 16CIF are defined in Recommendation H.263. For the H.261 algorithm, SQCIF is any active picture size less than QCIF, filled out by a black border, and coded in the QCIF format. For all these formats, the pixel aspect ratio is the same as that of the CIF format.

NOTE – The resulting picture aspect ratio for H.263 SQCIF is different from the other formats.

The video bit rate, picture format and algorithm options that can be accepted by the decoder are defined during the capability exchange using H.245. The encoder is free to transmit anything that is within the decoder capability set. The decoder should have the possibility to generate requests via H.245 for a certain mode, but the encoder is allowed to simply ignore these requests if they are not mandatory modes. Decoders which indicate capability for a particular algorithm option shall also be capable of accepting video bit streams which do not make use of that option.

H.323 terminals shall be capable of operating in asymmetric video bit rates, frame rates, and, if more than one picture resolution is supported, picture resolutions. For example, this will allow a CIF capable terminal to transmit QCIF while receiving CIF pictures.

When each video logical channel is opened, the maximum operating mode to be used on that channel is signalled to the receiver in the H.245 **OpenLogicalChannel** message. The maximum mode signalled includes maximum picture format, algorithm options, maximum codec bit rate, etc. The header within the video logical channel indicates which mode is actually used for each picture, within the stated maximum. For example, a video logical channel opened for CIF format may transmit CIF, QCIF, or SQCIF pictures, but not 4CIF or 16CIF. A video logical channel opened with only the **unrestrictedVector** and **arithmeticCoding** options may use neither, either, or both options, but shall not use options which were not signalled.

The video stream is formatted as described in Recommendation H.225.0.

6.2.4.1 Terminal-based continuous presence

H.323 terminals may receive more than one video channel, particularly for multipoint conferencing. In these cases, the H.323 terminal may need to perform a video mixing or switching function in order to present the video signal to the user. This function may include presenting the video from more than one terminal to the user. The H.323 terminal shall use H.245 simultaneous capabilities to indicate how many simultaneous video streams it is capable of decoding. The simultaneous capability of one terminal should not limit the number of video streams which are multicast in a conference (this choice is made by the MC).

6.2.5 Audio codec

All H.323 terminals shall have an audio codec. All H.323 terminals shall be capable of encoding and decoding speech according to Recommendation G.711. All terminals shall be capable of transmitting and receiving A-law and μ-law. A terminal may optionally be capable of encoding and decoding speech using Recommendations G.722, G.728, G.729, MPEG 1 audio, and G.723. The audio algorithm used by the encoder shall be derived during the capability exchange using H.245. The H.323 terminal should be capable of asymmetric operation for all audio capabilities it has declared within the same capability set, e.g. it should be able to send G.711 and receive G.728 if it is capable of both.

The audio stream is formatted as described in Recommendation H.225.0.

The H.323 terminal may optionally send more than one audio channel at the same time, for example, to allow two languages to be conveyed.

Audio packets should be delivered to the transport layer periodically at an interval determined by the audio codec Recommendation in use (audio frame interval). The delivery of each audio packet shall occur no later than 5 ms after a whole multiple of the audio frame interval, measured from delivery of the first audio frame (audio delay jitter). Audio coders capable of further limiting their audio delay jitter may so signal using the H.245 **maximumDelayJitter** parameter of the **h2250Capability**

structure contained within a terminal capability set message, so that receivers may optionally reduce their jitter delay buffers. This is not the same as the RTCP interarrival jitter field.

NOTE – The testing point for the maximum delay jitter is at the input to network transport layer. Network stack, network, driver, and interface card jitter is not included.

6.2.5.1 Audio mixing

H.323 terminals may receive more than one audio channel, particularly for multipoint conferencing. In these cases, the H.323 terminal may need to perform an audio mixing function in order to present a composite audio signal to the user. The H.323 terminal shall use H.245 simultaneous capabilities to indicate how many simultaneous audio streams it is capable of decoding. The simultaneous capability of one terminal should not limit the number of audio streams which are multicast in a conference.

6.2.5.2 Maximum audio-video transmit skew

To allow H.323 terminals to appropriately set their receive buffer(s) size, H.323 terminals shall transmit the **h2250MaximumSkewIndication** message to indicate the maximum skew between the audio and video signals as delivered to the network transport. **h2250MaximumSkewIndication** shall be sent for each pair of associated audio and video logical channels. This is not required for audio only or hybrid conferences Lip synchronization, if desired, shall be achieved via use of time-stamps.

6.2.6 Receive path delay

Receive path delay includes delay added to a media stream in order to maintain synchronization and to account for network packet arrival jitter. Media streams may optionally be delayed in the receiver processing path to maintain synchronization with other media streams. Further, a media stream may optionally be delayed to allow for network delays which cause packet arrival jitter. An H.323 terminal shall not add delay for this purpose in its transmitting media path.

Intermediate processing points such as MCUs or Gateways may alter the video and audio time tag information, and shall transmit appropriately modified audio and video time tags and sequence numbers, reflecting their transmitted signals. Receiving endpoints may add appropriate delay in the audio path to achieve lip synchronization.

6.2.7 Data channel

One or more data channels are optional. The data channel may be unidirectional or bidirectional depending on the requirements of the data application.

Recommendation T.120 is the default basis of data interoperability between an H.323 terminal and other H.323, H.320, or H.310 terminals. Where any optional data application is implemented using one or more of the ITU-T Recommendations which can be negotiated via H.245, the equivalent T.120 application, if any, shall be one of those provided. A terminal that provides far-end camera control using H.281 and H.224 is not required to also support a T.120 far-end camera control protocol.

Note that non-standard data applications (**dataApplicationCapability** = **non-standard** application, **dataProtocolCapability** = **non-standard**) and Transparent User Data (**dataApplicationCapability** = **userData** application, **dataProtocolCapability** = **transparent**) may be used whether the equivalent T.120 application is provided or not.

T.120 capability shall be signalled using **dataApplicationCapability** = **t120** application, **dataProtocolCapability** = **separateLANStack**.

The Data channel is formatted as described in Recommendation H.225.0.

6.2.7.1 T.120 data channels

There are two ways of establishing a T.120 connection within the context of an H.323 call. The first is the establishment of the T.120 connection during an H.323 call as an inherent part of the call, using the procedures of H.245 for opening logical channels. The second is the establishment of the T.120 connection prior to placing the H.323 call.

In the case where the H.323 call is established first, the normal call set-up procedures of 8.1 are followed. The capability exchange takes place, and a bidirectional logical channel shall be opened for the T.120 connection according to the normal H.245 procedures indicating that a new connection shall be created as described below.

The opening of a bidirectional logical channel for T.120 may be initiated by either entity sending **openLogicalChannel,** and then following the bidirectional logical channel procedures of Recommendation H.245.

To actually open the logical channel, the initiating entity shall send an **openLogicalChannel** message indicating that a T.120 data channel is to be opened in the **forwardLogicalChannelParameters** as well as in the **reverseLogicalChannelParameters**. The initiator may decide whether or not to include a transport address in the **openLogicalChannel** message. If the peer (the responder) accepts this logical channel, it shall confirm the opening of the logical channel using **openLogicalChannelAck**. In the **openLogicalChannelAck**, the responder shall include a transport address to be used for set-up of the T.120 connection if it did not receive a transport address from the initiator. Otherwise, the transport address shall be absent. In both cases, the transport address for the T.120 connection shall be carried in the **separateStack** parameter.

The entity transmitting the transport address shall be prepared to accept a T.120 connection on this transport address. The entity receiving the transport address shall initiate a T.120 connection set-up using the previously received transport address.

In both the **openLogicalChannel** and the **openLogicalChannelAck** messages, the **associateConference** parameter shall be set to **false**.

Recommendation T.120 shall follow the procedures of Recommendation T.123 for the protocol stack indicated in the **dataProtocolCapability** except that the transport addresses as described above shall be employed for connection set-up. The winner of the H.245 master/slave determination process shall have the option of being the upper node in the T.120 connection.

NOTE – The T.120 operation after completion of the connection set-up is beyond the scope of this Recommendation.

In the case where the T.120 connection is established first, the H.323 call is placed following the normal call set-up procedures of 8.1. The capability exchange takes place, and it is desired to associate the T.120 connection with the H.323 call. The procedures of Recommendation H.245 are used to open a bidirectional logical channel for T.120 data as described below.

The opening of a bidirectional logical channel for T.120 may be initiated by either entity by sending **openLogicalChannel**, and then following the bidirectional logical channel procedures of Recommenation H.245. The initiator of the set-up should include identification information for the already existing T.120 connection in the **openLogicalChannel** message to indicate to the peer which T.120 connection (if there are several) is to be associated.

To actually open the logical channel, the initiating entity shall send an **openLogicalChannel** message for a bidirectional logical channel indicating that a T.120 data channel is to be opened in the **forwardLogicalChannelParameters** as well as in the **reverseLogicalChannelParameters**. If the peer accepts this logical channel, it shall return an **openLogicalChannelAck** message to the

initiator in which it may include its local identification for the transport connection. In both messages, the **associateConference** parameter shall be set to **true**.

As identification information, the local (dynamically instantiated) transport address of the initial transport connection of the T.120 connection should be provided in the **separateStack** parameter. In addition, the **externalReference** parameter may be used to provide further information on which T.120 connection is to be associated.

If any of this identification information is not available to the initiator, it may omit the respective value(s).

NOTE – If the transport address is not specified and the two endpoints share more than one T.120 connection it may be ambiguous for the recipient which T.120 connection is referred to. Implications of this ambiguity and steps to avoid it are for further study.

6.2.8 H.245 control function

The H.245 Control Function uses the H.245 Control Channel to carry end-to-end control messages governing operation of the H.323 entity, including capabilities exchange, opening and closing of logical channels, mode preference requests, flow control messages, and general commands and indications.

H.245 signalling is established between two endpoints: an endpoint and an MC, or an endpoint and a Gatekeeper. The endpoint shall establish exactly one H.245 Control Channel for each call that the endpoint is participating in. This channel shall use the messages and procedures of Recommendation H.245. Note that a terminal, MCU, Gateway, or Gatekeeper may support many calls, and thus many H.245 Control Channels. The H.245 Control Channel shall be carried on logical channel 0. Logical channel 0 shall be considered to be permanently open from the establishment of the H.245 Control Channel until the termination of this channel. The normal procedures for opening and closing logical channels shall not apply to the H.245 Control Channel.

Recommendation H.245 specifies a number of independent protocol entities which support endpoint-to-endpoint signalling. A protocol entity is specified by its syntax (messages), semantics, and a set of procedures which specify the exchange of messages and the interaction with the user. H.323 endpoints shall support the syntax, semantics, and procedures of the following protocol entities:

- Master/slave determination.
- Capability Exchange.
- Logical Channel Signalling.
- Bidirectional Logical Channel Signalling.
- Close Logical Channel Signalling.
- Mode Request.
- Round Trip Delay Determination.
- Maintenance Loop Signalling.

General commands and indications shall be chosen from the message set contained in Recommendation H.245. In addition, other command and indication signals may be sent which have been specifically defined to be transferred in-band within video, audio or data streams (see the appropriate Recommendation to determine if such signals have been defined).

H.245 messages fall into four categories: Request, Response, Command, and Indication. Request and Response messages are used by the protocol entities. Request messages require a specific action by the receiver, including an immediate response. Response messages respond to a corresponding request. Command messages require a specific action, but do not require a response. Indication

messages are informative only, and do not require any action or response. H.323 terminals shall respond to all H.245 commands and requests as specified in Annex A, and shall transmit indications reflecting the state of the terminal.

H.323 terminals shall be capable of parsing all H.245 **MultimediaSystemControlMessage** messages, and shall send and receive all messages needed to implement required functions and those optional functions which are supported by the terminal. Annex A contains a table showing which H.245 messages are mandatory, optional, or forbidden for H.323 terminals. H.323 terminals shall send the **functionNotSupported** message in response to any unrecognized request, response, or command messages that it receives.

An H.245 indication, **userInputIndication**, is available for transport of user input alphanumeric characters from a keypad or keyboard, equivalent to the DTMF signals used in analogue telephony, or SBE number messages in Recommendation H.230. This may be used to manually operate remote equipment such as voice mail or video mail systems, menu-driven information services, etc. H.323 terminals shall support the transmission of user input characters 0-9, "*", and "#". Transmission of other characters is optional.

NOTE – If the encryption procedures of this Recommendation are in use, the H.245 Control Channel will not be encrypted. Users are therefore cautioned regarding the carriage of user data in the H.245 Control Channel, the use of non-standard messages, and the confidentiality risk from traffic analysis of the H.245 Control Channel.

Three H.245 request messages conflict with RTCP control packets. The H.245 **videoFastUpdatePicture**, **videoFastUpdateGOB** and **videoFastUpdateMB** requests should be used instead of the RTCP control packets Full Intra Request (FIR) and Negative Acknowledgement (NACK). The ability to accept FIR and NACK is signalled during the H.245 capability exchange.

6.2.8.1 Capabilities exchange

Capabilities exchange shall follow the procedures of Recommendation H.245, which provides for separate receive and transmit capabilities, as well as a method by which the terminal may describe its ability to operate in various combinations of modes simultaneously.

Receive capabilities describe the terminal's ability to receive and process incoming information streams. Transmitters shall limit the content of their transmitted information to that which the receiver has indicated it is capable of receiving. The absence of a receive capability indicates that the terminal cannot receive (is a transmitter only).

Transmit capabilities describe the terminal's ability to transmit information streams. Transmit capabilities serve to offer receivers a choice of possible modes of operation, so that the receiver may request the mode which it prefers to receive. The absence of a transmit capability indicates that the terminal is not offering a choice of preferred modes to the receiver (but it may still transmit anything within the capability of the receiver).

The transmitting terminal assigns each individual mode the terminal is capable of operating in a number in a **capabilityTable**. For example, G.723 audio, G.728 audio, and CIF H.263 video would each be assigned separate numbers.

These capability numbers are grouped into **alternativeCapabilitySet** structures. Each **alternativeCapabilitySet** indicates that the terminal is capable of operating in exactly one mode listed in the set. For example, an **alternativeCapabilitySet** listing {G.711, G.723, G.728} means that the terminal can operate in any one of those audio modes, but not more than one.

These **alternativeCapabilitySet** structures are grouped into **simultaneousCapabilities** structures. Each **simultaneousCapabilities** structure indicates a set of modes the terminal is capable of using simultaneously. For example, a **simultaneousCapabilities** structure containing the two

alternativeCapabilitySet structures {H.261, H.263} and {G.711, G.723, G.728} means that the terminal can operate either of the video codecs simultaneously with any one of the audio codecs. The **simultaneousCapabilities** set { {H.261}, {H.261, H.263}, {G.711, G.723, G.728} } means the terminal can operate two video channels and one audio channel simultaneously: one video channel per H.261, another video channel per either H.261 or H.263, and one audio channel per either G.711, G.723, or G.728.

NOTE – The actual capabilities stored in the **capabilityTable** are often more complex than presented here. For example, each H.263 capability indicates details including the ability to support various picture formats at given minimum picture intervals, and the ability to use optional coding modes. For a complete description, see Recommendation H.245.

The terminal's total capabilities are described by a set of **capabilityDescriptor** structures, each of which is a single **simultaneousCapabilities** structure and a **capabilityDescriptorNumber**. By sending more than one **capabilityDescriptor**, the terminal may signal dependencies between operating modes by describing different sets of modes which it can simultaneously use. For example, a terminal issuing two **capabilityDescriptor** structures, one { {H.261, H.263}, {G.711, G.723, G.728} } as in the previous example, and the other { {H.262}, {G.711} }, means the terminal can also operate the H.262 video codec, but only with the low-complexity G.711 audio codec.

Terminals may dynamically add capabilities during a communication session by issuing additional **capabilityDescriptor** structures, or remove capabilities by sending revised **capabilityDescriptor** structures. All H.323 terminals shall transmit at least one **capabilityDescriptor** structure.

Non-standard capabilities and control messages may be issued using the **nonStandardParameter** structure defined in Recommendation H.245. Note that while the meaning of non-standard messages is defined by individual organizations, equipment built by any manufacturer may signal any non-standard message, if the meaning is known.

Terminals may re-issue capability sets at any time, according to the procedures of Recommendation H.245.

6.2.8.2 Logical channel signalling

Each logical channel carries information from a transmitter to one or more receivers, and is identified by a logical channel number which is unique for each direction of transmission.

Logical channels are opened and closed using the **openLogicalChannel** and **closeLogicalChannel** messages and procedures of Recommendation H.245. When a logical channel is opened, the **openLogicalChannel** message fully describes the content of the logical channel, including media type, algorithm in use, any options, and all other information needed for the receiver to interpret the content of the logical channel. Logical channels may be closed when no longer needed. Open logical channels may be inactive, if the information source has nothing to send.

Most logical channels in this Recommendation are unidirectional, so asymmetrical operation, in which the number and type of information streams is different in each direction of transmission, is allowed. However, if a receiver is capable only of certain symmetrical modes of operation, it may send a receive capability set that reflects its limitations, except where noted elsewhere in this Recommendation. Terminals may also be capable of using a particular mode in only one direction of transmission. Certain media types, including data protocols such as T.120, inherently require a bidirectional channel for their operation. In such cases a pair of unidirectional logical channels, one in each direction, may be opened and associated together to form a bidirectional channel using the bidirectional channel opening procedures of Recommendation H.245. Such pairs of associated channels need not share the same logical channel number, since logical channel numbers are independent in each direction of transmission.

Logical channels shall be opened using the following procedure:

The initiating terminal shall send an **OpenLogicalChannel** message as described in Recommendation H.245. If the logical channel is to carry a media type using RTP (audio or video), the **OpenLogicalChannel** message shall include the **mediaControlChannel** parameter containing the transport address for the reverse RTCP channel.

The responding terminal shall respond with an **OpenLogicalChannelAck** message as described in Recommendation H.245. If the logical channel is to carry a media type using RTP, the **OpenLogicalChannelAck** message shall include both the **mediaTransportChannel** parameter containing the RTP transport address for the media channel and the **mediaControlChannel** parameter containing the transport address for the forward RTCP channel.

Media types (such as T.120 data) which do not use RTP/RTCP shall omit the **mediaControlChannel** parameters.

If a corresponding reverse channel is opened for a given existing RTP session (identified by the RTP **sessionID**), the **mediaControlChannel** transport addresses exchanged by the **OpenLogicalChannel** process shall be identical to those used for the forward channel. Should a collision occur where both ends attempt to establish conflicting RTP sessions at the same time, the master endpoint shall reject the conflicting attempt as described in Recommendation H.245. The rejected **OpenLogicalChannel** attempt may then be retried at a later time.

6.2.8.3 Mode preferences

Receivers may request transmitters to send a particular mode using the H.245 **requestMode** message, which describes the desired mode. Transmitters should comply if possible.

An endpoint receiving the **multipointModeCommand** from the MC shall then comply with all **requestMode** commands, if they are within its capability set. Note that in a decentralized conference, as in a centralized conference, all terminal **requestMode** commands are directed to the MC. The MC may grant the request or not; the basis for this decision is left to the manufacturer.

6.2.8.4 Master-slave determination

The H.245 Master-slave determination procedures are used to resolve conflicts between two endpoints which can both be the MC for a conference, or between two endpoints which are attempting to open a bidirectional channel. In this procedure, two endpoints exchange random numbers in the H.245 **masterSlaveDetermination** message, to determine the master and slave endpoints. H.323 endpoints shall be capable of operating in both master and slave modes. The endpoints shall set **terminalType** to the value specified in Table 1 below and set **statusDeterminationNumber** to a random number in the range 0 to 2^{24}–1. Only one random number shall be chosen by the endpoint for each call, except in the case of identical random numbers, as described in Recommendation H.245.

Table 1/H.323 – H.323 terminal types for H.245 master-slave determination

TerminalType Value Table	H.323 Entity			
Feature set	**Terminal**	**Gateway**	**Gatekeeper**	**MCU**
Entity with No MC	50	60	NA	NA
Entity contains an MC but no MP	70	80	120	160
Entity contains MC with data MP	NA	90	130	170
Entity contains MC with data and audio MP	NA	100	140	180
Entity contains MC with data, audio, and video MP	NA	110	150	190

The Active MC in a conference shall use a value of 240. Cascaded MCs are for further study.

If a single H.323 entity can take part in multiple calls, then the value used for **terminalType** in the master-slave determination process shall be based on the features that the H.323 entity has assigned or will assign to the call in which it is being signalled.

An MC that is already acting as an MC shall always remain the active MC. Therefore, once an MC has been selected as the active MC in a conference, it shall use the Active MC value for all subsequent connections to the conference.

If no MC is active and the entities are of the same type, then the H.323 entity with the highest feature set (as shown in Table 1) shall win the master-slave determination. If no MC is active and the entities are of different types, then an MC that is located in an MCU shall have priority over an MC that is located in a Gatekeeper, which shall have priority over an MC that is located in a Gateway, which in turn shall have priority over an MC located in a terminal.

If an H.323 entity can be associated with two or more of the classifications shown in Table 1, then it should use the highest value for which it qualifies.

6.2.8.5 Timer and counter values

All timers defined in Recommendation H.245 should have periods of at least as long as the maximum data delivery time allowed by the data link layer carrying the H.245 Control Channel, including any retransmissions.

The H.245 retry counter N100 should be at least 3.

Procedures relating to H.245 protocol error handling are covered in 8.6.

6.2.9 RAS signalling function

The RAS signalling function uses H.225.0 messages to perform registration, admissions, bandwidth changes, status, and disengage procedures between endpoints and Gatekeepers. The RAS Signalling Channel is independent from the Call Signalling Channel and the H.245 Control Channel. H.245 open logical channel procedures are not used to establish the RAS Signalling Channel. In LAN environments that do not have a Gatekeeper, the RAS Signalling Channel is not used. In LAN environments which contain a Gatekeeper (a Zone), the RAS Signalling Channel is opened between the endpoint and the Gatekeeper. The RAS Signalling Channel is opened prior to the establishment of any other channels between H.323 endpoints. This channel is described in detail in clause 7.

6.2.10 Call signalling function

The call signalling function uses H.225.0 call signalling to establish a connection between two H.323 endpoints. The Call Signalling Channel is independent from the RAS Channel and the H.245 Control Channel. H.245 open logical channel procedures are not used to establish the Call Signalling

Channel. The Call Signalling Channel is opened prior to the establishment of the H.245 Channel and any other logical channels between H.323 endpoints. In systems that do not have a Gatekeeper, the Call Signalling Channel is opened between the two endpoints involved in the call. In systems which contain a Gatekeeper, the Call Signalling Channel is opened between the end point and the Gatekeeper, or between the endpoints themselves as chosen by the Gatekeeper. This channel is described in detail in clause 7.

6.2.11 H.225.0 layer

Logical channels of video, audio, data or control information are established according to the procedures of Recommendation H.245. Logical channels are unidirectional, and are independent in each direction of transmission. Some logical channels, such as for data, may be bidirectional, and are associated through the bidirectional open logical channel procedure of Recommendation H.245. Any number of logical channels of each media type may be transmitted, except for the H.245 Control Channel of which there shall be one per call. In addition to the logical channels, H.323 endpoints use two signalling channels for call control, and Gatekeeper related functions. The formatting used for these channels shall conform to Recommendation H.225.0.

6.2.11.1 Logical channel numbers

Each logical channel is identified by a Logical Channel Number (LCN), in the range 0 to 65 535, which serves only to associate logical channels with the transport connection. Logical channel numbers are selected arbitrarily by the transmitter, except that logical channel 0 shall be permanently assigned to the H.245 Control Channel. The actual Transport Address that the transmitter shall transmit to, shall be returned by the receiver in the **openLogicalChannelAck** message.

6.2.11.2 Logical channel bit rate limits

A logical channel's bandwidth shall have an upper limit specified by the minimum of the endpoint's transmit capability (if present) and the receiving endpoint's receive capability. Based on this limit, an endpoint shall open a logical channel at a bit rate at or below this upper limit. A transmitter shall transmit an information stream within the logical channel at any bit rate at or below the open logical channel bit rate. The limit applies to the information streams which are the content of the logical channel(s), not including RTP headers, RTP payload headers and LAN headers and other overhead.

H.323 endpoints shall obey to the **flowControlCommand** message of H.245, which commands a limit to the bit rate of a logical channel or the aggregate bit rate of all logical channels. H.323 endpoints that want to limit the bit rate of a logical channel, or the aggregate bit rate of all logical channels should send the **flowControlCommand** message to the transmitting endpoint.

When the terminal has no information to send in a given channel, the terminal shall send no information. Fill data shall not be sent on the LAN in order to maintain a specific data rate.

6.3 Gateway characteristics

The Gateway shall provide the appropriate translation between transmission formats (for example H.225.0 to/from H.221) and between communications procedures (for example H.245 to/from H.242). The Gateway shall also perform call set-up and clearing on both the LAN side and the SCN side. Translation between video, audio, and data formats may also be performed in the Gateway. In general, the purpose of the Gateway (when not operating as an MCU) is to reflect the characteristics of a LAN endpoint to an SCN endpoint, and the reverse, in a transparent fashion.

An H.323 endpoint may communicate with another H.323 endpoint on the same LAN directly and without involving a Gateway. The Gateway may be omitted if communications with SCN terminals (terminals not on the LAN) is not required. It may also be possible for a terminal on one segment of

the LAN to call out through one Gateway and back onto the LAN through another Gateway in order to bypass a router or a low bandwidth link.

The Gateway has the characteristics of an H.323 Terminal or MCU on the LAN, and of the SCN terminal or MCU on the SCN. The choice of terminal or MCU is left to the manufacturer. The Gateway provides the necessary conversion between the different terminal types. Note that the Gateway may initially operate as a terminal, but later using H.245 signalling begin to operate as an MCU for the same call that was initially point-to-point. Gatekeepers are aware of which terminals are Gateways since this is indicated when the terminal/Gateway registers with the Gatekeeper.

A Gateway which passes T.120 data between the SCN and the LAN may contain a T.120 MCS Provider which connects the T.120 MCS Providers on the LAN to the T.120 MCS Providers on the SCN.

Four examples of an H.323 Gateway are shown in Figure 5. The diagrams show the H.323 terminal or MCU function, the SCN terminal or MCU function, and the conversion function. The H.323 terminal function has the characteristics described in 6.2. The H.323 MCU function has the characteristics described in 6.5. The Gateway appears to the other H.323 terminals on the LAN as one or more H.323 terminals, or an H.323 MCU. It communicates with the other H.323 terminals using the procedures in this Recommendation.

The SCN terminal or MCU function has the characteristics described in the appropriate Recommendation (H.310, H.320, H.321, H.322, H.324, V.70, GSTN or ISDN speech only terminals). The Gateway appears to the terminals on the SCN as one or more of the same terminal types or MCUs. It communicates to another terminal on the SCN using the procedures described in the appropriate Recommendation for that terminal. SCN signalling procedures are beyond the scope of this Recommendation, including such topics as whether the H.323 Gateway appears as a terminal or a network to the SCN. Note that a Gateway may convert H.323 directly to H.324 or H.310 without going to H.320.

Gateways supporting interworking with speech only terminals on GSTN or ISDN should generate and detect DTMF signals corresponding to H.245 **userInputIndications** for 0-9, *, and #.

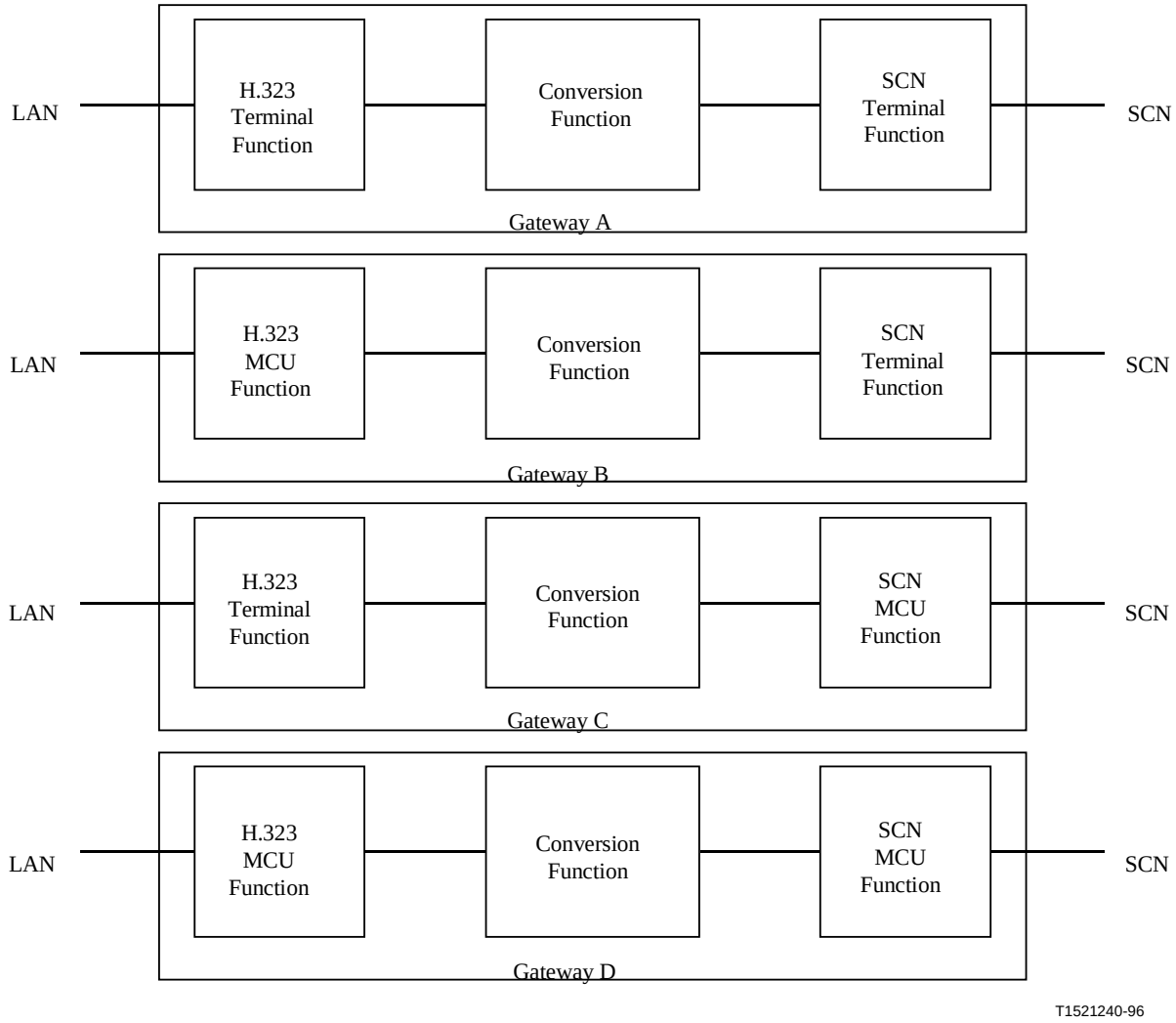


Figure 5/H.323 – H.323 gateway configurations

The conversion function provides the necessary conversion of transmission format, control, audio, video, and/or data streams between the different terminal Recommendations. At a minimum, the Gateway shall provide a conversion function for the transmission format, call set-up signals and procedures, and communications control signals and procedures. When required, the Gateway shall provide for H.242 to H.245 conversion. The Gateway performs the appropriate conversion between the H.225.0 Call Signalling and the SCN signalling system (Q.931, Q.2931, etc.). The conversion between Q.931 messages on the LAN and Q.931 messages on the SCN is described in Appendix A.

All Q.931 signalling received by the Gateway from an ISDN endpoint and not applicable to the Gateway should be passed through to the LAN endpoint, and vice versa. This signalling includes Q.932 and Q.950-Series messages. This will allow H.323 endpoints to implement the Supplementary Services defined in those Recommendations. The handling of other SCN call signalling systems is for further study.

This Recommendation describes the connection of one H.323 terminal on the LAN to one external terminal on the SCN through the Gateway. The actual number of H.323 terminals that can communicate through the Gateway is not subject to standardization. Similarly, the number of SCN connections, number of simultaneous independent conferences, audio/video/data conversion functions, and inclusion of multipoint functions is left to the manufacturer. If the Gateway includes an MCU function on the LAN side, that function shall be an H.323 MCU on the LAN side. If the

Gateway includes an MCU function on the SCN side, it may appear as an H.231/H.243 MCU, or as an MCU for H.310 or H.324 systems (these MCUs are indicated as for further study in the respective Recommendations) on the SCN side.

A Gateway may be connected via the SCN to other Gateways to provide communication between H.323 terminals which are not on the same LAN.

Equipment which provides transparent interconnection between LANs without using H.320 (such as routes and remote dial in units) are not Gateways as defined within the scope of this Recommendation.

6.4 Gatekeeper characteristics

The Gatekeeper, which is optional in an H.323 system, provides call control services to the H.323 endpoints. More than one Gatekeeper may be present and communicate with each other in an unspecified fashion. The Gatekeeper is logically separate from the endpoints, however, its physical implementation may coexist with a terminal, MCU, Gateway, MC, or other non-H.323 LAN device.

When it is present in a system, the Gatekeeper shall provide the following services:

- Address Translation – The Gatekeeper shall perform alias address to Transport Address translation. This should be done using a translation table which is updated using the Registration messages described in clause 7. Other methods of updating the translation table are also allowed.
- Admissions Control – The Gatekeeper shall authorize LAN access using ARQ/ACF/ARJ H.225.0 messages. This may be based on call authorization, bandwidth, or some other criteria which is left to the manufacturer. It may also be a null function which admits all requests.
- Bandwidth Control – The Gatekeeper shall support BRQ/BRJ/BCF messages. This may be based on bandwidth management. It may also be a null function which accepts all requests for bandwidth changes.
- Zone Management – The Gatekeeper shall provide the above functions for terminals, MCUs, and Gateways which have registered with it as described in 7.2.

The Gatekeeper may also perform other optional functions such as:

- Call Control Signalling – The Gatekeeper may choose to complete the call signalling with the endpoints and may process the call signalling itself. Alternatively, the Gatekeeper may direct the endpoints to connect the Call Signalling Channel directly to each other. In this manner, the Gatekeeper can avoid handling the H.225.0 call control signals. The Gatekeeper may have to act as the network as defined in Recommendation Q.931 in order to support supplementary services. This operation is for further study.
- Call Authorization – Through the use of the H.225.0 signalling, the Gatekeeper may reject calls from a terminal due to authorization failure. The reasons for rejection may include, but are not limited to, restricted access to/from particular terminals or Gateways, and restricted access during certain periods of time. The criteria for determining if authorization passes or fails is outside the scope of this Recommendation.
- Bandwidth Management – Control of the number of H.323 terminals permitted simultaneous access to the LAN. Through the use of the H.225.0 signalling, the Gatekeeper may reject calls from a terminal due to bandwidth limitations. This may occur if the Gatekeeper determines that there is not sufficient bandwidth available on the network to support the call. The criteria for determining if bandwidth is available is outside the scope of this Recommendation. Note that this may be a null function, i.e. all terminals are granted access.

This function also operates during an active call when a terminal requests additional bandwidth.

- Call Management – For example, the Gatekeeper may maintain a list of ongoing H.323 calls. This information may be necessary to indicate that a called terminal is busy, and to provide information for the Bandwidth Management function.
- Gatekeeper management information data structure – For further study.
- Bandwidth reservation for terminals not capable of this function – For further study.
- Directory services – For further study.

In order to support ad hoc Multipoint Conferences, the Gatekeeper may choose to receive the H.245 Control Channels from the two terminals in a point-to-point conference. When the conference switches to a multipoint conference, the Gatekeeper can redirect the H.245 Control Channel to an MC. The Gatekeeper need not process the H.245 signalling; it only needs to pass it between the terminals or the terminals and the MC.

LANs which contain Gateways should also contain a Gatekeeper in order to translate incoming E.164 addresses into Transport Addresses.

H.323 entities that contain a Gatekeeper shall have a mechanism to disable the internal Gatekeeper so that when there are multiple H.323 entities that contain a Gatekeeper on a LAN, the H.323 entities can be configured into the same Zone.

6.5 Multipoint controller characteristics

The MC provides control functions to support conferences between three or more endpoints in a multipoint conference. The MC carries out the capabilities exchange with each endpoint in a multipoint conference. The MC sends a capability set to the endpoints in the conference indicating the operating modes in which they may transmit. The MC may revise the capability set that it sends to the terminals as a result of terminals joining or leaving the conference, or for other reasons.

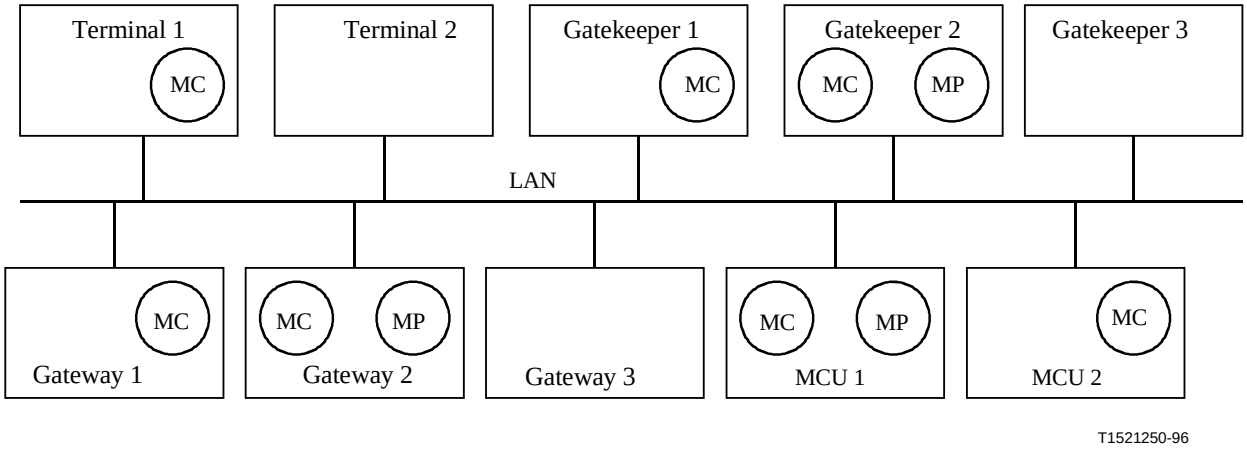
In this manner, the MC determines the Selected Communication Mode (SCM) for the conference. The SCM may be common for all endpoints in the conference. Alternatively, some endpoints may have a different SCM than other endpoints in the conference. The manner in which the MC determines an SCM is not within the scope of this Recommendation.

As part of multipoint conference set-up, an endpoint will become connected to an MC on its H.245 Control Channel. This connection may occur:

– via an explicit connection with an MCU;
– via an implicit connection to the MC within a Gatekeeper;
– via an implicit connection to the MC within another terminal or Gateway in the multipoint conference;
– via an implicit connection through a Gatekeeper to an MCU.

The choice of conference mode (e.g. decentralized or centralized) occurs after connection with the MC using H.245 signalling. The choice of conference mode may be limited by the capability of the endpoints or the MC.

The MC may be located within a Gatekeeper, Gateway, terminal, or MCU. See Figure 6.



NOTE – Gateway, Gatekeeper and MCU can be a single device.

Figure 6/H.323 – Possible locations of MC and MP in H.323 system

An MC within a terminal is not callable. It can be included in the call in order to process the H.245 signalling to support ad hoc multipoint conferences. In this case, there may be no distinction between the MC and the H.245 Control Function (see 6.2.8) of the terminal. Communications between them is outside the scope of this Recommendation.

An MC located with the Gatekeeper is not callable; however, an MCU located with a Gatekeeper is callable. An MCU located with a Gatekeeper functions as an independent MCU. An MC located with a Gatekeeper may be used to support ad hoc multipoint conferences when the Gatekeeper receives the H.245 Control Channels from the endpoints. In this manner, the Gatekeeper can route the H.245 Control Channels to the MC at the start of the call or when the conference switches to multipoint.

The Gateway can function as a terminal or an MCU. When functioning as a terminal, the Gateway may contain an MC. This has the same characteristics as described above for an MC within a terminal.

An MCU always contains an MC. The MCU is callable and the MC processes the H.245 Control Channel from all of the endpoints.

When two or more endpoints are in a conference, the endpoints shall use the master slave resolution procedure of Recommendation H.245 to determine the MC that will control the conference.

After the capability exchange and master/slave determination, the MC may first assign a terminal number to a new endpoint using the **terminalNumberAssign**. The MC shall then notify the other endpoints of the new endpoint in the conference using **terminalJoinedConference**. The new endpoint may request a list of other endpoints in the conference using the **terminalListRequest**.

6.6 Multipoint processor characteristics

The MP receives audio, video and/or data streams from the endpoints involved in a centralized or hybrid multipoint conference. The MP processes these media streams and returns them to the endpoints.

Communications between the MC and the MP are not subject to standardization.

The MP may process one or more media stream types. When the MP processes video, it shall process the video algorithms and formats as described in 6.2.4. When the MP processes audio, it

shall process the audio algorithms as described in 6.2.5. When the MP processes data, it shall process data streams as described in 6.2.7.

An MP which processes video shall provide either video switching or video mixing. Video switching is the process of selecting the video that the MP outputs to the terminals from one source to another. The criteria used to make the switch may be determined through detection of a change in speaker (sensed by the associated audio level) or through H.245 control. Video mixing is the process of formatting more than one video source into the video stream that the MP outputs to the terminals. An example of video mixing is combining four source pictures into a two-by-two array in the video output picture. The criteria for which sources and how many are mixed is determined by the MC until other controls are defined. The use of the T.120-Series Recommendations for these control functions is for further study.

An MP which processes audio shall prepare N-audio outputs from M-audio inputs by switching, mixing, or a combination of these. Audio mixing requires decoding the input audio to linear signals (PCM or analogue), performing a linear combination of the signals and recoding the result to the appropriate audio format. The MP may eliminate or attenuate some of the input signals in order to reduce noise and other unwanted signals. Each audio output may have a different mix of input signals providing for private conversations. The terminals shall assume that their audio is not present in the audio stream returned to them. Terminal removal of its own audio from the MP audio output is for further study.

An MP which processes T.120 data shall be capable of acting as a non-leaf MCS provider and should be capable of acting as the Top MCS Provider. An MP may also process non-standard data, transparent user data and/or other types of data.

The MP may provide algorithm and format conversion, allowing terminals to participate in a conference at different SCMs.

The MP is not callable, the MCU which it is a part of is callable. The MP terminates and sources the media channels.

6.7 Multipoint control unit characteristics

The MCU is an endpoint which provides support for multipoint conferences. The MCU shall consist of an MC and zero or more MPs. The MCU uses H.245 messages and procedures to implement features similar to those found in Recommendation H.243.

A typical MCU that supports centralized multipoint conferences consists of an MC and an audio, video and data MP. A typical MCU that supports decentralized multipoint conferences consists of an MC and a data MP supporting Recommendation T.120. It relies on decentralized audio and video processing.

The LAN side of a Gateway may be an MCU. A Gatekeeper may also include an MCU. In either case they are independent functions that happen to be co-located.

The MCU shall be callable by other endpoints using the procedures of clause 8.

6.8 Multipoint capability

6.8.1 Centralized multipoint capability

All terminals shall have centralized multipoint capability. A Gateway which appears as a terminal on the LAN shall also have centralized multipoint capability. In this mode of operation they communicate with the MC of the MCU in a point-to-point manner on the control channel and with the MP on the audio, video and data channels. In this mode, the MC performs H.245 multipoint

control functions, while the MP performs video switching or mixing, audio mixing, and T.120 multipoint data distribution. The MP transmits the resulting video, audio and data streams back to the terminals. The MP may have the capability to convert between different audio, video and data formats and bit rates, allowing the terminals to participate in the conference using different communications modes.

The MCU may use multicast to distribute the processed video if the terminals in the conference can receive multicast transmissions. Multicast distribution of processed audio is for further study.

This mode is signalled by the following H.245 capabilities: **centralizedControl**, **centralizedAudio**, **centralizedVideo**, and **centralizedData**.

6.8.2 Decentralized multipoint capability

If the terminals have decentralized multipoint capability, they communicate with the MC of an MCU, Gateway, Gatekeeper, or terminal in a point-to-point mode on the H.245 Control Channel and optionally with an MP on data channels. The terminals shall have the capability to multicast their audio and video channels to all other endpoints in the conference. The MC may control which terminal or terminals are actively multicasting audio and/or video (for example by using the **flowControlCommand** on either channel).

The terminals receive multicast video channels and select one or more of the available channels for display to the user. The terminals receive the multicast audio channels and perform an audio mixing function in order to present a composite audio signal to the user.

The MC may provide conference control functions such as chair control, video broadcast and video selection. This shall be done by receiving H.245 from a terminal and then sending the appropriate control to other terminals to enable or disable their video multicast. T.120 commands may optionally provide the same functions.

This mode is signalled by the following H.245 capabilities: **centralizedControl**, **decentralizedAudio**, **decentralizedVideo**, and **centralizedData**.

6.8.3 Hybrid multipoint – Centralized audio

If the terminals and MCU have hybrid multipoint-centralized audio capability, they may use distributed multipoint for video and centralized multipoint for audio. In this mode, the terminals communicate with the MC in a point-to-point mode on the H.245 Control Channel and optionally with an MP on data channels.

The terminals shall have the capability to multicast their video channels to all other endpoints in the conference. The MC may control which terminal or terminals are actively multicasting video. The terminals receive multicast video channels and select one or more of the available channels for display to the user.

All of the terminals in the conference transmit their audio channels to the MP. The MP performs the audio mixing function and outputs the resulting audio streams to the terminals. The MP may produce an exclusive audio sum for each terminal in the conference. Multicast distribution of processed audio is for further study.

This mode is signalled by the following H.245 capabilities: **centralizedControl**, **centralizedAudio**, **decentralizedVideo**, and **centralizedData**.

6.8.4 Hybrid multipoint – Centralized video

If the terminals and MCU have hybrid multipoint-centralized video capability, they may use distributed multipoint for audio and centralized multipoint for video. In this mode, the terminals

communicate with the MC in a point-to-point mode on the H.245 Control Channel and optionally with an MP on data channels.

The terminals shall have the capability to multicast their audio channels to all other endpoints in the conference. The MC may control which terminal or terminals are actively multicasting audio. The terminals receive multicast audio channels and perform a mixing function in order to present a composite audio signal to the user.

All of the terminals in the conference transmit their video channels to the MP. The MP performs the video switching, mixing, or format conversion functions and outputs the resulting video streams to the terminals. The MP may produce an exclusive video stream for each terminal in the conference, or it may multicast a video stream to all participating terminals, in order to minimize the bandwidth used on the LAN.

This mode is signalled by the following H.245 capabilities: **centralizedControl**, **decentralizedAudio**, **centralizedVideo**, and **centralizedData**.

6.8.5 Establishment of common mode

The MC shall coordinate a common communications mode between the terminals in the multipoint conference. The MC may force terminals into a particular common mode of transmission (as allowed by their capability sets) by sending to the terminal a receive capability set listing only the desired mode of transmission, or the MC may rely on **multipointModeCommand** and mode preference commands to enforce mode symmetry. The latter approach should be used since it allows the terminals to know the full range of conference capabilities available that can be requested.

If the MCU has the capability to convert audio and/or video formats, it may not be necessary to force all terminals into the same communications mode.

6.8.6 Multipoint rate matching

Since the terminals on each link in a multipoint configuration may attempt to operate at different bit rates, the MC shall send H.245 **flowControlCommand** messages to limit the transmitted bit rates to those which can be sent to receivers.

6.8.7 Multipoint lip synchronization

An MP which is providing audio mixing in either the Centralized or Hybrid multipoint conferences shall modify the time tags of the audio and video streams, taking into account its own time base, in order to maintain audio and video synchronization. Further, when the MP processes the audio and/or video to generate a new stream sourced from the MP, the MP shall generate its own sequence numbers in the audio and video packets.

When mixing audio, the MP should synchronize each of the incoming audio streams to its own timing, mix the audio streams, and then shall generate a new audio stream based on its own timing with its own sequence numbers. If the MP is also switching video, the switched stream shall have its original time-stamp replaced with the MP time base to synchronize it with the mixed audio stream and shall have a new sequence number representing the stream from the MP.

In the case of distributed multipoint conferences, the receiving terminal may be able to maintain lip synchronization by aligning the selected video stream and its associated audio by using the RTP time tags. Alignment of the other audio streams may not be necessary. If multiple video streams are displayed, the associated audio streams should be aligned.

It may not be possible to guarantee lip synchronization in hybrid multipoint conferences.

6.8.8 Multipoint encryption

In a centralized multipoint configuration, the MP is considered to be a trusted entity. Each port of the MP decrypts the information streams from each of the H.323 terminals and encrypts the information streams to each terminal in accordance with 10.1. Operation of an untrusted MCU is for further study.

Encryption in decentralized and hybrid multipoint conferences is for further study.

6.8.9 Cascading multipoint control units

The multipoint control function may be distributed between several MCU entities. Such operations are for further study.

7 Call signalling

Call signalling is the messages and procedures used to establish a call, request changes in bandwidth of the call, get status of the endpoints in the call, and disconnect the call. Call signalling uses messages defined in Recommendation H.225.0 and the procedures described in clause 8. This clause describes some call signalling concepts.

7.1 Addresses

7.1.1 LAN address

Each H.323 entity shall have at least one LAN Address. This address uniquely identifies the H.323 entity on the LAN. Some entities may share a LAN Address (i.e. a terminal and a co-located MC). This address is specific to the LAN environment in which the endpoint is located. Different LAN environments may have different LAN Address formats.

An endpoint may use different LAN addresses for different channels within the same call.

7.1.2 TSAP identifier

For each LAN Address, each H.323 entity may have several TSAP Identifiers. These TSAP Identifiers allow multiplexing of several channels sharing the same LAN Address.

Endpoints have one well-known TSAP Identifier defined: the Call Signalling Channel TSAP Identifier. Gatekeepers have one well-known TSAP Identifier defined: the RAS Channel TSAP Identifier and one well-known multicast address defined: Discovery Multicast Address. These are defined in Appendix IV/H.225.0.

Endpoints and H.323 entities should use dynamic TSAP Identifiers for the H.245 Control Channel, Audio Channels, Video Channels, and Data Channels. The Gatekeeper should use a dynamic TSAP Identifier for Call Signalling Channels. The RAS Channels and Signalling Channels may be redirected to dynamic TSAP Identifiers during the registration procedure.

7.1.3 Alias address

An endpoint may also have one or more alias addresses associated with it. The alias addresses provide an alternate method of addressing the endpoint. These address include E.164 addresses (network access number, telephone number, etc.), H.323 IDs (name, e-mail like address, etc.), and any others defined in Recommendation H.225.0. Alias addresses shall be unique within a Zone. Gatekeepers, MCs, and MPs shall not have alias addresses.

When there is no Gatekeeper in the system, the calling endpoint shall address the called endpoint directly using the Call Signalling Channel Transport Address of the called endpoint. When there is a

Gatekeeper in the system, the calling endpoint may address the called endpoint by its Call Signalling Channel Transport Address, or alias address. The latter shall be translated into a Call Signalling Channel Transport Address by the Gatekeeper.

The called endpoint's E.164 address may consist of an optional access code followed by the E.164 address. The access code consists of n digits from the set of 0 to 9, *, and #. The number of digits and their meaning is left to the discretion of the manufacturer. One purpose of such an access code might be to request access to a Gateway. The Gatekeeper may alter this address prior to sending it to the destination.

The H.323 ID consists of a string of ISO/IEC 10646-1 characters as defined in Recommendation H.225.0. It may be a user name, e-mail name, or other identifier.

An endpoint may have more than one alias address (including more than one of the same type) which is translated to the same Transport Address. The endpoint's alias addresses shall be unique within a Zone.

7.2 Registration, Admissions and Status (RAS) channel

The RAS Channel shall be used to carry messages used in the Gatekeeper discovery and endpoint registration processes which associate an endpoint's alias address with its Call Signalling Channel Transport Address. The RAS Channel shall be an unreliable channel.

7.2.1 Gatekeeper discovery

Gatekeeper discovery is the process an endpoint uses to determine which Gatekeeper to register with. This may be done manually or automatically. Manual discovery relies on methods outside the scope of this Recommendation to determine which Gatekeeper an endpoint is associated with. The endpoint is configured with the Transport Address of the associated Gatekeeper. For example, it may be entered at endpoint configuration, or it may be entered into an initialization file. In this way, the endpoint knows *a priori* which Gatekeeper it is associated with. The endpoint can now register with that Gatekeeper.

The Automatic method allows the endpoint-Gatekeeper association to change over time. The endpoint may not know who its Gatekeeper is, or may need to identify another Gatekeeper due to a failure. This may be done through auto discovery. Auto discovery allows for lower administrative overhead in configuring individual endpoints and additionally allows replacement of an existing Gatekeeper without manually reconfiguring all of the affected endpoints.

The endpoint may multicast (or use other methods as described in Appendix IV/H.225.0 a Gatekeeper Request (GRQ) message, asking "Who is my Gatekeeper?". This is sent to the Gatekeeper's well-known Discovery Multicast Address. One or more Gatekeepers may respond with the Gatekeeper Confirmation (GCF) message indicating "I can be your Gatekeeper.", and returns the Transport Address of the Gatekeeper's RAS Channel. If a Gatekeeper does not want the endpoint to register to it, it shall return Gatekeeper Reject (GRJ). See Figure 7. If more than one Gatekeeper responds, the endpoint may choose the Gatekeeper it wants to use. At this point, the endpoint knows which Gatekeeper to register with. The endpoint can now register with that Gatekeeper.

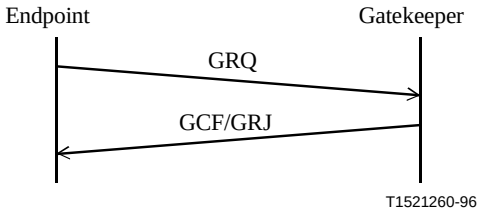


Figure 7/H.323 – Auto Discovery

If no Gatekeeper responds within a timeout, the endpoint may retry the GRQ. An endpoint shall not send a GRQ within 5 s after sending a previous one. If no response is received, the endpoint may use the manual discovery method.

If at any time an endpoint determines it has an invalid registration with its Gatekeeper, it must re-discover its Gatekeeper. The invalid registration may be detected by either receiving an RRJ message from a Gatekeeper in response to an RRQ from the endpoint, or not receiving any response to an RRQ from the endpoint within a time-out.

The GRQ may be repeated periodically (i.e. at endpoint power-up), so the Gatekeeper shall be able to handle multiple requests from the same endpoint.

7.2.2 Endpoint registration

Registration is the process by which an endpoint joins a Zone, and informs the Gatekeeper of its Transport Address and alias addresses. As part of their configuration process, all endpoints shall register with the Gatekeeper identified through the discovery process. Registration shall occur before any calls are attempted and may occur periodically as necessary (for example, at endpoint power-up).

An endpoint shall send a Registration Request (RRQ) to a Gatekeeper. This is sent to the Gatekeeper's RAS Channel Transport Address. The endpoint has the LAN Address of the Gatekeeper from the Gatekeeper discovery process and uses the well-known RAS Channel TSAP Identifier. The Gatekeeper shall respond with either a Registration Confirmation (RCF) or a Registration Reject (RRJ). See Figure 8. An endpoint shall only register with a single Gatekeeper.

The RRQ may be repeated periodically (i.e. at terminal power-up), so the Gatekeeper shall be able to handle multiple requests from the same endpoint. If a Gatekeeper receives an RRQ having the same alias address and the same Transport Address as a previous RRQ, it shall respond with RCF. If a Gatekeeper receives an RRQ having the same alias address as a previous RRQ and a different Transport Address, it should reject the registration indicating a duplicate registration. If the Gatekeeper receives an RRQ having the same Transport Address as a previous RRQ and a different alias address, it should replace the translation table entries. The Gatekeeper may have a method to authenticate these changes which is for further study.

The Gatekeeper shall ensure that each alias address translates uniquely to a single Transport Address. Ambiguous registrations shall be rejected by the Gatekeeper. The Gatekeeper may reject the registration for other reasons, such as changes in discovery or security issues.

If the endpoint does not include an alias address in the RRQ message, the Gatekeeper may assign one. The Gatekeeper shall return the assigned alias address to the terminal in the RCF message.

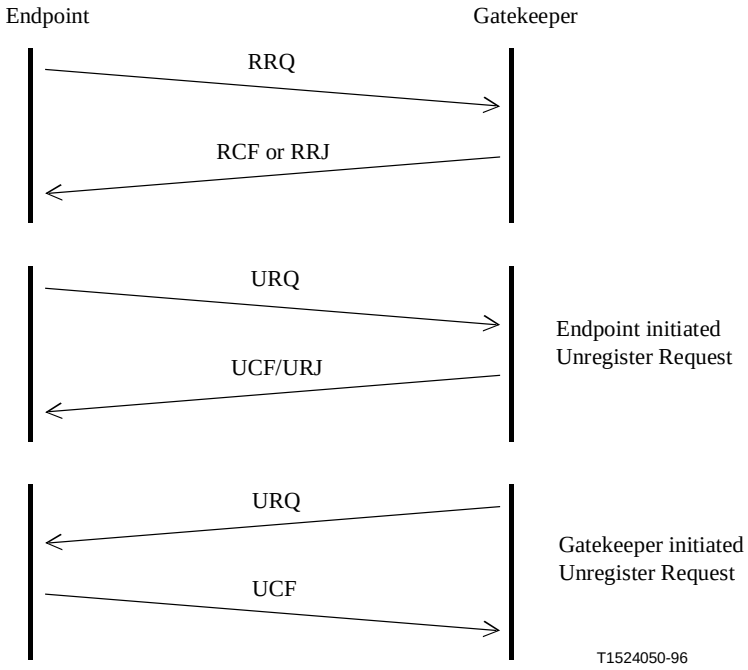


Figure 8/H.323 – Registration

An endpoint may cancel its registration by sending an Unregister Request (URQ) message to the Gatekeeper. The Gatekeeper shall respond with an Unregister Confirmation (UCF) message. This allows endpoints to change the alias address associated with its Transport Address, or vice versa. If the endpoint was not registered with the Gatekeeper, it shall return an Unregister Reject (URJ) message to the endpoint.

A Gatekeeper may cancel the registration of an endpoint by sending an Unregister Request (URQ) message to the endpoint. The endpoint shall respond with an Unregister Confirmation (UCF) message. The endpoint shall re-register with a Gatekeeper prior to initiating any calls. This may require the endpoint to register with a new Gatekeeper.

An endpoint which is not registered with a Gatekeeper is called an unregistered endpoint. This type of endpoint does not request admission permission from a Gatekeeper and so cannot participate in admissions control, bandwidth control, address translation and other functions performed by the Gatekeeper.

7.2.3 Endpoint location

An endpoint or Gatekeeper which has an alias address for an endpoint and would like to determine its Transport Address may issue a Location Request (LRQ) message. This message may be sent to a specific Gatekeeper, or may be multicast like the GRQ message. This is sent to the Gatekeeper's well-known RAS Channel TSAP Identifier, or if multicast, is sent to the Gatekeeper's well-known Discovery Multicast Address. The Gatekeeper with which the requested endpoint is registered, shall respond with the Location Confirmation (LCF) message containing the Transport Address of the endpoint's Call Signalling Channel or the endpoint's Gatekeeper's Call Signalling Channel. All Gatekeepers with which the requested endpoint is not registered, shall return Location Reject (LRJ) if they received the LRQ on the RAS Channel. Any Gatekeeper with which the requested endpoint is not registered, shall not respond to the LRQ, if it received the LRQ on the Discovery Multicast address.

7.2.4 Admissions, bandwidth change, status, disengage

The RAS Channel is also used for the transmission of Admissions, Bandwidth Change, Status, and Disengage messages. These messages take place between an endpoint and a Gatekeeper and are used to provide admissions control and bandwidth management functions. The detailed use of these messages is described in clause 8.

The Admissions Request (ARQ) message specifies the requested Call Bandwidth. This is an upper limit on the aggregate bit rate for all transmitted and received, audio and video channels excluding any RTP headers, RTP payload headers, LAN headers, and other overhead. Data and control channels are not included in this limit. The Gatekeeper may reduce the requested Call Bandwidth in the Admissions Confirm (ACF) message. An endpoint shall assure that the aggregate bit rate, averaged over one second, for all transmitted and received, audio and video channels is at or below the Call Bandwidth. An endpoint or the Gatekeeper may attempt to modify the Call Bandwidth during a call using the Bandwidth Change Request (BRQ) message.

7.3 Call signalling channel

The Call Signalling Channel shall be used to carry H.225.0 call control messages. The Call Signalling channel shall be a reliable channel.

In networks that do not contain a Gatekeeper, call signalling messages are passed directly between the calling and called endpoints using the Call Signalling Transport Addresses. In these networks, it is assumed that the calling endpoint knows the Call Signalling Transport Address of the called endpoint and thus can communicate directly.

In networks that do contain a Gatekeeper, the initial admission message exchange takes place between the calling endpoint and the Gatekeeper using the Gatekeeper's RAS Channel Transport Address. Within the initial admissions message exchange, the Gatekeeper indicates in the ACF message whether to send the call signalling directly to the other endpoint or to route it through the Gatekeeper. The call signalling messages are sent to either the endpoint's Call Signalling Transport Address or the Gatekeeper's Call Signalling Transport Address.

Recommendation H.225.0 specifies the mandatory Q.931 messages that are used for call signalling in this Recommendation. Clause 8 specifies the procedures for using them.

7.3.1 Call signalling channel routing

Call signalling messages may be passed in two ways. The first method is Gatekeeper Routed Call Signalling (see Figure 9). In this method, call signalling messages are routed through the Gatekeeper between the endpoints. The second method is Direct Endpoint Call Signalling (see Figure 10). In this method, the call signalling messages are passed directly between the endpoints. The choice of which methods is used is made by the Gatekeeper.

Both methods use the same kinds of connections for the same purposes, and the same messages. Admissions messages are exchanged on RAS channels with the Gatekeeper, followed by an exchange of call signalling messages on a Call Signalling channel. This is then followed by the establishment of the H.245 Control Channel. The actions of the Gatekeeper in response to the admission messages determine which call model is used this is not under the control of the endpoint, although the endpoint can specify a preference.

For the Gatekeeper Routed method, the Gatekeeper may choose to close the Call Signalling Channel after the call set-up is completed, or it may choose to keep it open for the duration of the call to support supplementary services. Only the Gatekeeper shall close the Call Signalling Channel and it should not be closed when a Gateway is involved in the call.

The symmetrical signalling method of Annex D/Q.931 shall be used for all mandatory call signalling procedures. This does not address the role that a Gateway might play on the SCN side using Q.931 or other call signalling protocols.

The Gatekeeper Clouds in Figures 9 through 12 contain one or more Gatekeepers which may or may not communicate with each other. The endpoints may be connected to the same Gatekeeper or to different Gatekeepers.

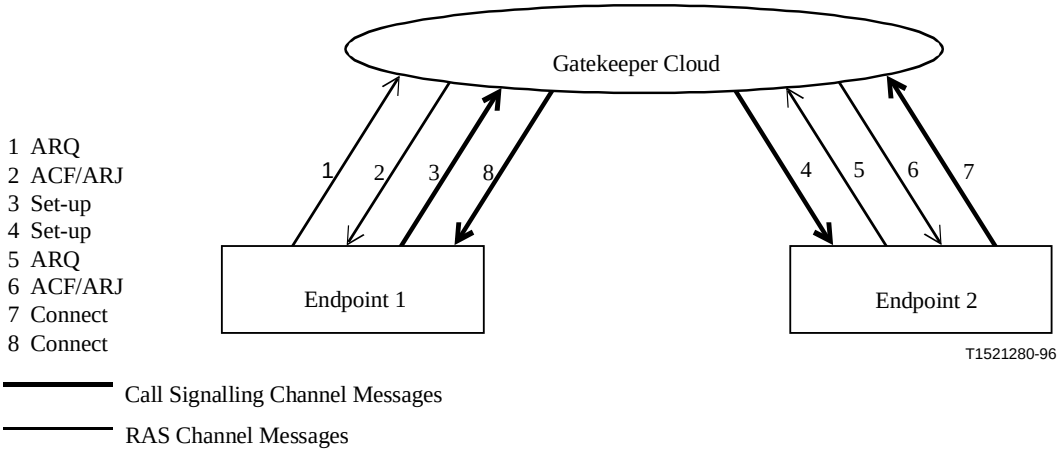


Figure 9/H.323 – Gatekeeper routed call signalling

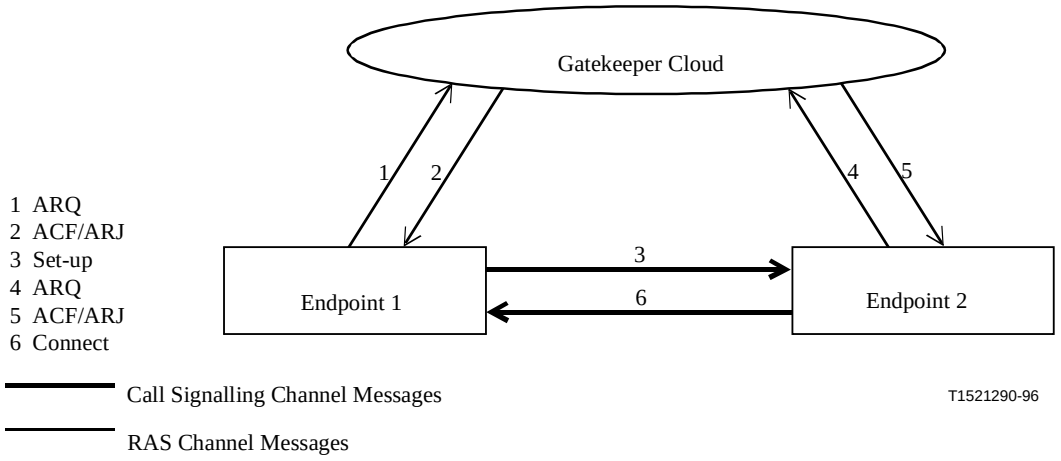


Figure 10/H.323 – Direct endpoint call signalling

7.3.2 Control channel routing

When Gatekeeper Routed call signalling is used, there are two methods to route the H.245 Control Channel. In the first method, the H.245 Control Channel is established directly between the endpoints. See Figure 11. This method is for further study. In the second method, the H.245 Control Channel is routed between the endpoints through the Gatekeeper. See Figure 12. This method allows the Gatekeeper to redirect the H.245 Control Channel to an MC when an ad hoc multipoint conference switches from a point-to-point conference to a multipoint conference. This choice is

made by the Gatekeeper. When Direct Endpoint call signalling is used, the H.245 Control Channel can only be connected directly between the endpoints.

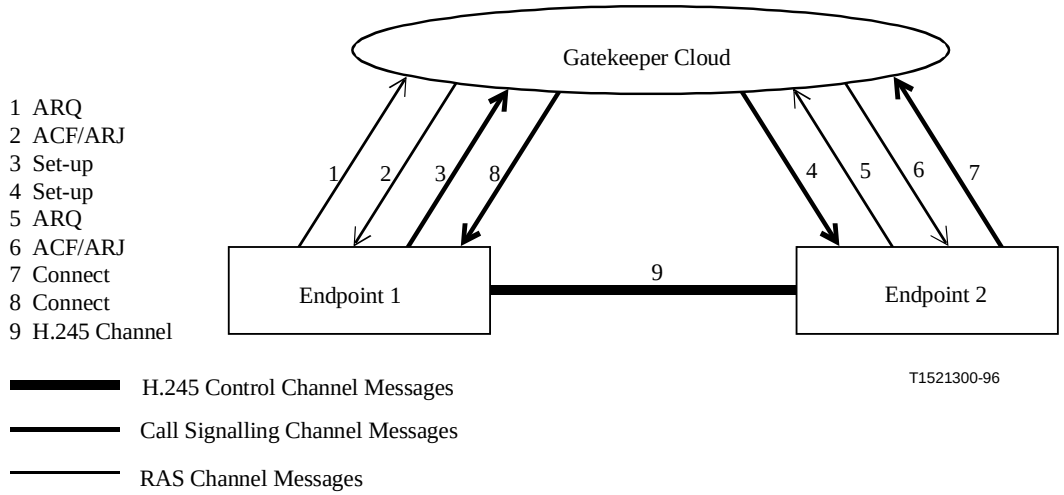


Figure 11/H.323 – Direct H.245 control channel connection between endpoints

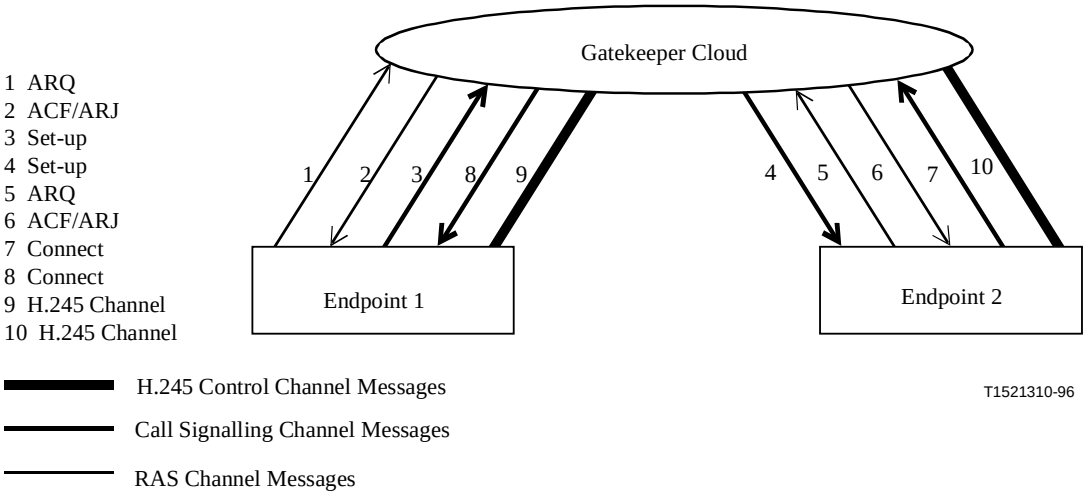


Figure 12/H.323 – Gatekeeper routed H.245 control

7.4 Call reference value

All call signalling and RAS messages contain a Call Reference Value (CRV). Refer to Recommendation H.225.0. This is used to associate all messages related to the same call. The same CRV shall be used in all admissions, call set-up, supplementary service, bandwidth change, and call termination messages relating to the same call. A new CRV shall be used for new calls. A second call from an endpoint to invite another endpoint into the same conference shall use a new CRV. The CRV is not the same as the Conference ID (CID). The CRV relates messages within the same call, the CID relates calls within the same conference.

7.5 Conference ID and Conference Goal

The Conference ID (CID) is a unique non-zero value created by the originating endpoint and passed in various H.225.0 messages, encoded with CID octet zero first. The CID identifies the conference with which the message is associated. The CID shall be formed from 16 octets as follows:

CID octet	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	N5	N4	N3	N2	N1	N0	C1	C0	H1	AV	M1	M0	L3	L2	L1	L0

where an index of 0 (e.g. N0) refers to the lowest order octet of the respective value (e.g. N5:N0):

N5:0 is 48 bits of physical LAN address, if available.

C1:0 is 16 bits of counter incremented per conference if the clock cannot be guaranteed to be monotonic.

H1, A, M1:0, L3:0 is low order 60 bits of 100 nanosecond clock since October 15, 1582 local timezone. The assignment of bits is as follows:

H1		A		M1	M0	L3	L2	L1	L0
59	52	51	48	47	32	31			0

V is 4 bits of version number = 0001 placed in the lower order 4 bits of CID octet 6.

The **conferenceGoal** indicates the intention of the call. Choices are:

Create – To create a new conference;

Join – To join an existing conference; and

Invite – To invite a new endpoint into an existing conference.

8 Call signalling procedures

The provision of the communication is made in the following steps:

– Phase A: Call set-up (see 8.1).

– Phase B: Initial communication and capability exchange (see 8.2).

– Phase C: Establishment of audiovisual communication (see 8.3).

– Phase D: Call Services (see 8.4).

– Phase E: Call termination (see 8.5).

8.1 Phase A – Call set-up

Call set-up takes place using the call control messages defined in Recommendation H.225.0 according to the call control procedures defined below. Requests for bandwidth reservation should take place at the earliest possible phase.

If both the alias address and the transport address are specified, preference shall be given to the alias address.

8.1.1 Basic call set-up – Neither endpoint registered

In the scenario shown in Figure 13 neither endpoint is registered to a Gatekeeper. The two endpoints communicate directly. Endpoint 1 (calling endpoint) sends the Set-up (1) message to the well-known Call Signalling Channel TSAP Identifier of endpoint 2. Endpoint 2 responds with the Connect (4) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling.

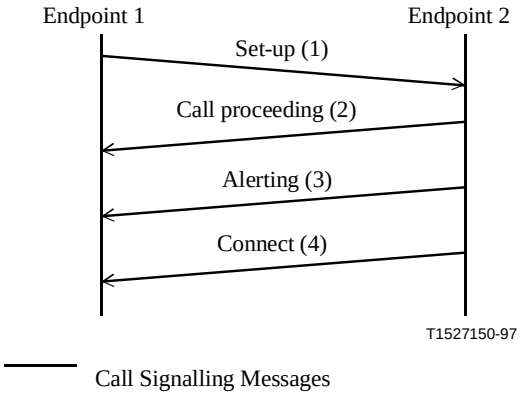


Figure 13/H.323 – Basic call set-up, no Gatekeepers

8.1.2 Both endpoints registered to the same Gatekeeper

In the scenario shown in Figure 14, both endpoints are registered to the same Gatekeeper, and the Gatekeeper has chosen Direct Call Signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with that Gatekeeper. The Gatekeeper shall return the Call Signalling Channel Transport Address of endpoint 2 (called endpoint) in the ACF. Endpoint 1 then sends the Set-up (3) message to endpoint 2 using that Transport Address. If endpoint 2 wishes to accept the call, it initiates an ARQ (5)/ACF (6) exchange with the Gatekeeper. It is possible that an ARJ (6) is received by endpoint 2, in which case it sends Release Complete to endpoint 1. Endpoint 2 responds with the Connect (4) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling.

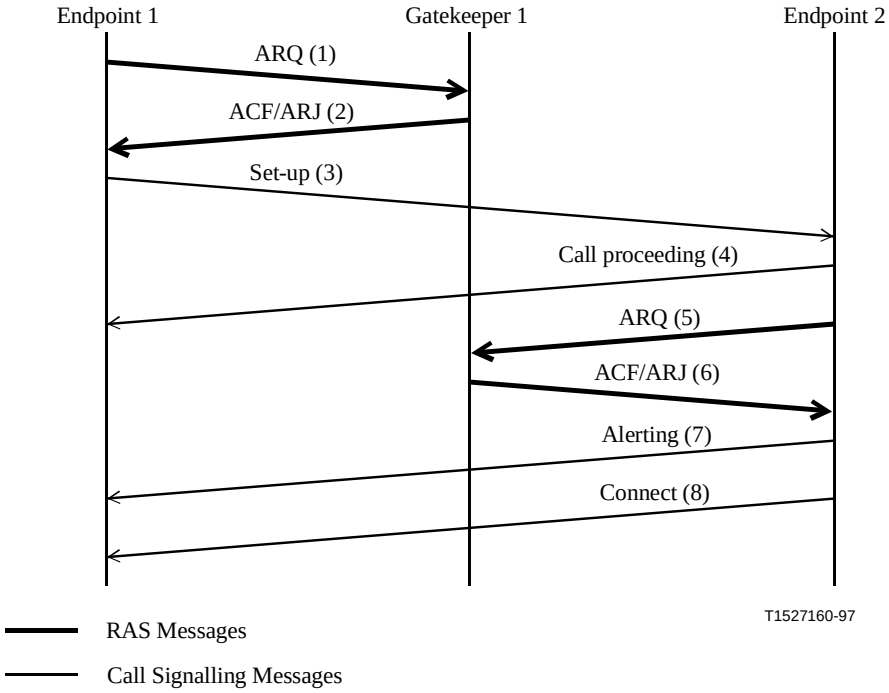


Figure 14/H.323 – Both endpoints registered, same Gatekeeper – Direct call signalling

In the scenario shown in Figure 15, both endpoints are registered to the same Gatekeeper, and the Gatekeeper has chosen to route the call signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with that Gatekeeper. The Gatekeeper shall return a Call Signalling Channel Transport Address of itself in the ACF. Endpoint 1 then sends the Set-up (3) message using that Transport Address. The Gatekeeper then sends the Set-up (4) message to endpoint 2. If endpoint 2 wishes to accept the call, it initiates an ARQ (6)/ACF (7) exchange with the Gatekeeper. It is possible that an ARJ (7) is received by endpoint 2, in which case it sends Release Complete to the Gatekeeper. Endpoint 2 responds with the Connect (9) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling. The Gatekeeper sends the Connect (10) message to endpoint 1 which may contain the endpoint 2 H.245 Control Channel Transport Address, or a Gatekeeper (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper chooses to route the H.245 Control Channel or not.

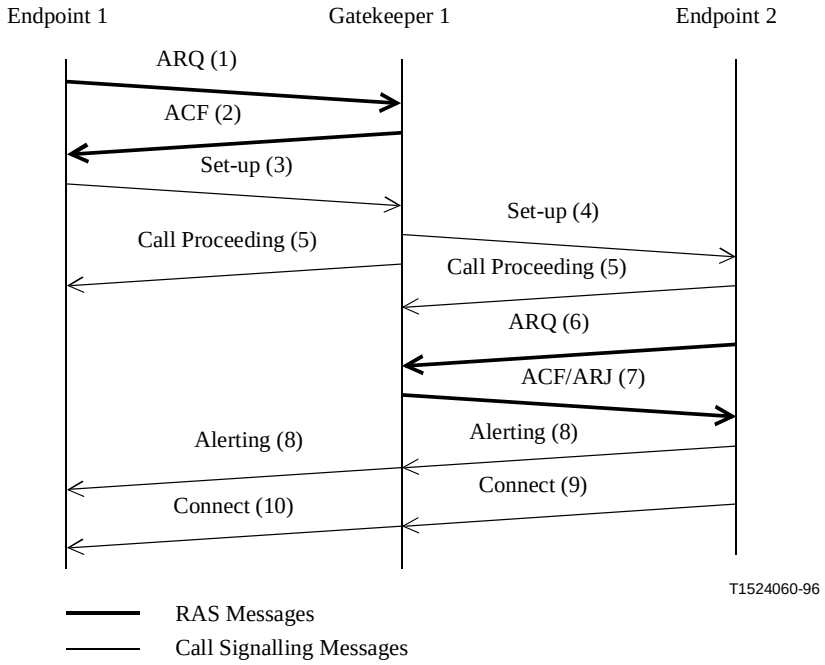


Figure 15/H.323 – Both endpoints registered, same Gatekeeper – Gatekeeper routed call signalling

8.1.3 Only calling endpoint has Gatekeeper

In the scenario shown in Figure 16, endpoint 1 (calling endpoint) is registered with a Gatekeeper, endpoint 2 (called endpoint) is not registered with a Gatekeeper, and the Gatekeeper has chosen Direct Call Signalling. Endpoint 1 initiates the ARQ (1)/ACF (2) exchange with the Gatekeeper. Endpoint 1 then sends the Set-up (3) message to endpoint 2 using the well-known Call Signalling Channel Transport Address. If endpoint 2 wishes to accept the call, it responds with the Connect (4) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling.

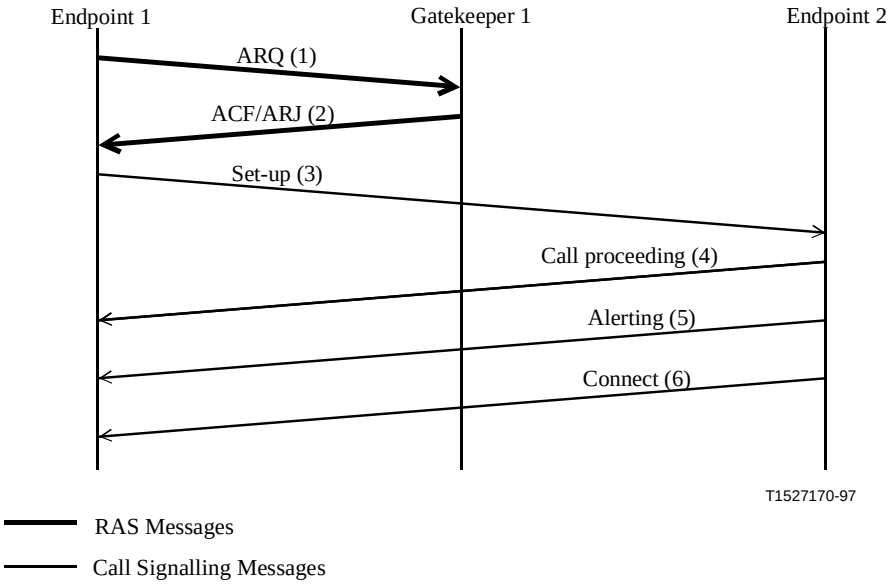


Figure 16/H.323 – Only calling endpoint registered – Direct call signalling

In the scenario shown in Figure 17, endpoint 1 (calling endpoint) is registered with a Gatekeeper, endpoint 2 (called endpoint) is not registered with a Gatekeeper, and the Gatekeeper has chosen to route the call signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with that Gatekeeper. The Gatekeeper shall return a Call Signalling Channel Transport Address of itself in the ACF (2). Endpoint 1 then sends the Set-up (3) message using that Transport Address. The Gatekeeper then sends the Set-up (4) message to the well-known Call Signalling Channel Transport Address of endpoint 2. If endpoint 2 wishes to accept the call, it responds with the Connect (7) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling. The Gatekeeper sends the Connect (9) message to endpoint 1 which may contain the endpoint 2 H.245 Control Channel Transport Address, or a Gatekeeper (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper chooses to route the H.245 Control Channel or not.

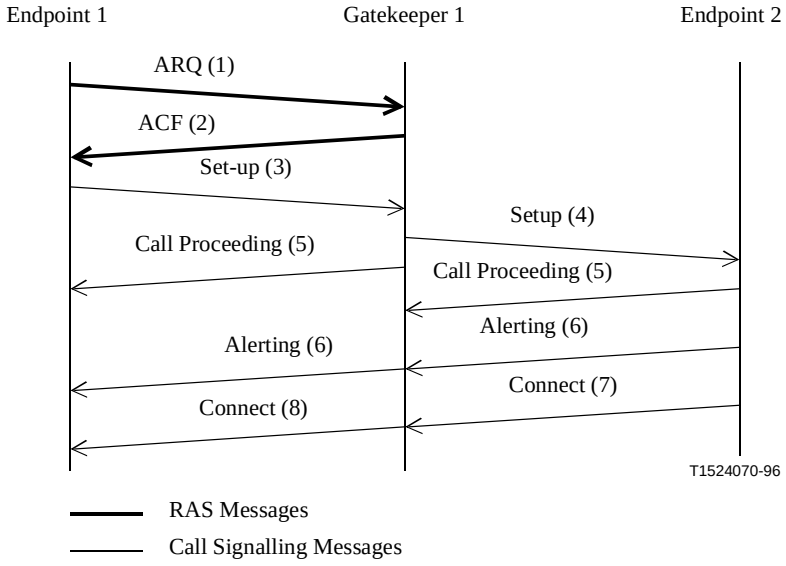


Figure 17/H.323 – Only calling endpoint registered – Gatekeeper routed call signalling

8.1.4 Only called endpoint has Gatekeeper

In the scenario shown in Figure 18, endpoint 1 (calling endpoint) is not registered with a Gatekeeper, endpoint 2 (called endpoint) is registered with a Gatekeeper, and the Gatekeeper has chosen Direct Call Signalling. Endpoint 1 sends the Set-up (1) message to endpoint 2 using the well-known Call Signalling Channel Transport Address. If endpoint 2 wishes to accept the call, it initiates an ARQ (3)/ACF (4) exchange with the Gatekeeper. It is possible that an ARJ (4) is received by endpoint 2, in which case it sends Release Complete to endpoint 1. Endpoint 2 responds with the Connect (6) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling.

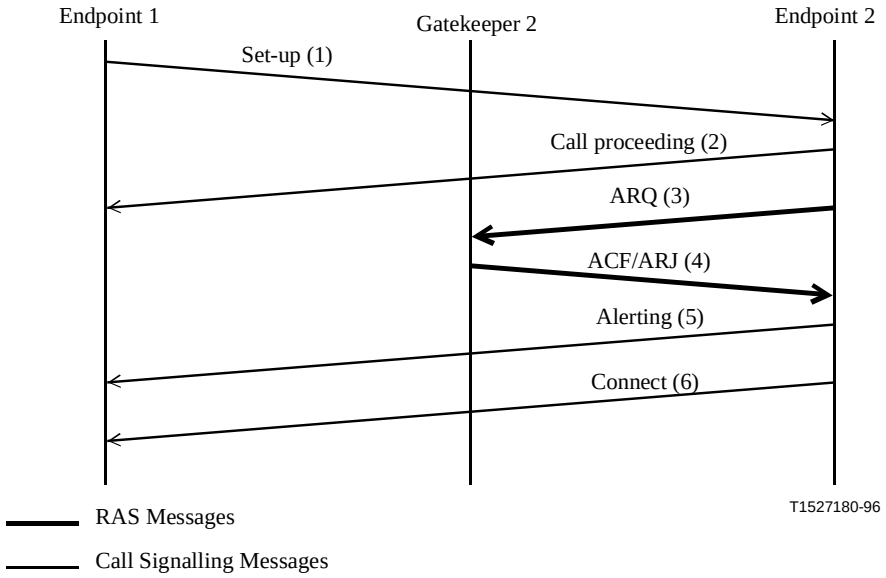


Figure 18/H.323 – Only called endpoint registered – Direct call signalling

In the scenario shown in Figure 19, endpoint 1 (calling endpoint) is not registered with a Gatekeeper, endpoint 2 (called endpoint) is registered with a Gatekeeper, and the Gatekeeper has chosen to route the call signalling. Endpoint 1 (calling endpoint) sends a Set-up (1) message to the well-known Call Signalling Channel Transport address of endpoint 2. If endpoint 2 wishes to accept the call, it initiates the ARQ (3)/ACF (4) exchange with that Gatekeeper. If acceptable, the Gatekeeper shall return a Call Signalling Channel Transport Address of itself in the ARJ (4) with a cause code of **routeCallToGatekeeper**. Endpoint 2 replies to endpoint 1 with a Facility (5) message containing the Call Signalling Transport Address of its Gatekeeper. Endpoint 1 then sends the Release Complete (6) message to endpoint 2. Endpoint 1 sends a Set-up (7) message to the Gatekeeper's Call Signalling Channel Transport Address. The Gatekeeper sends the Set-up (8) message to endpoint 2. Endpoint 2 initiates the ARQ (9)/ACF (10) exchange with that Gatekeeper. Endpoint 2 then responds with the Connect (12) message which contains its H.245 Control Channel Transport Address for use in H.245 signalling. The Gatekeeper sends the Connect (13) message to endpoint 1 which may contain the endpoint 2 H.245 Control Channel Transport Address, or a Gatekeeper (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper chooses to route the H.245 Control Channel or not.

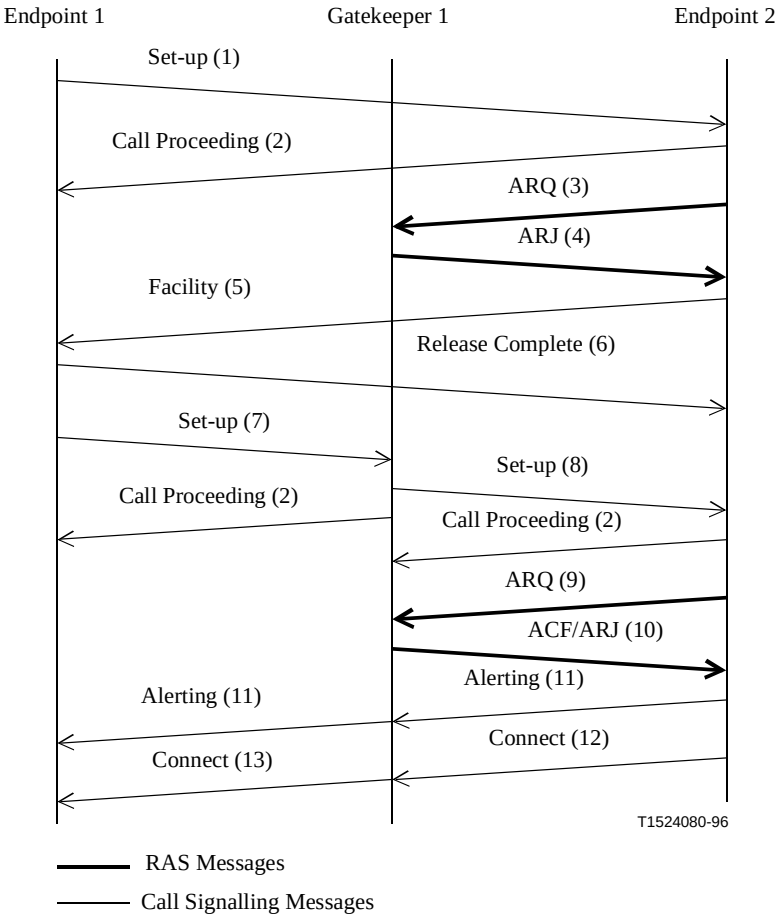


Figure 19/H.323 – Only called endpoint registered – Gatekeeper routed call signalling

8.1.5 Both endpoints registered to different Gatekeepers

In the scenario shown in Figure 20, both endpoints are registered to different Gatekeepers, and both Gatekeepers choose Direct Call Signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with Gatekeeper 1. Gatekeeper 1 may return the Call Signalling Channel Transport Address of endpoint 2 (called endpoint) in the ACF if Gatekeeper 1 has a method of communicating with Gatekeeper 2. Endpoint 1 then sends the Set-up (3) message to either the Transport Address returned by the Gatekeeper (if available) or to the well-known Call Signalling Channel Transport Address of endpoint 2. If endpoint 2 wishes to accept the call, it initiates an ARQ (5)/ACF (6) exchange with Gatekeeper 2. It is possible that an ARJ (6) is received by endpoint 2, in which case it sends Release Complete to endpoint 1. Endpoint 2 responds with the Connect (8) message which contains an H.245 Control Channel Transport Address for use in H.245 signalling.

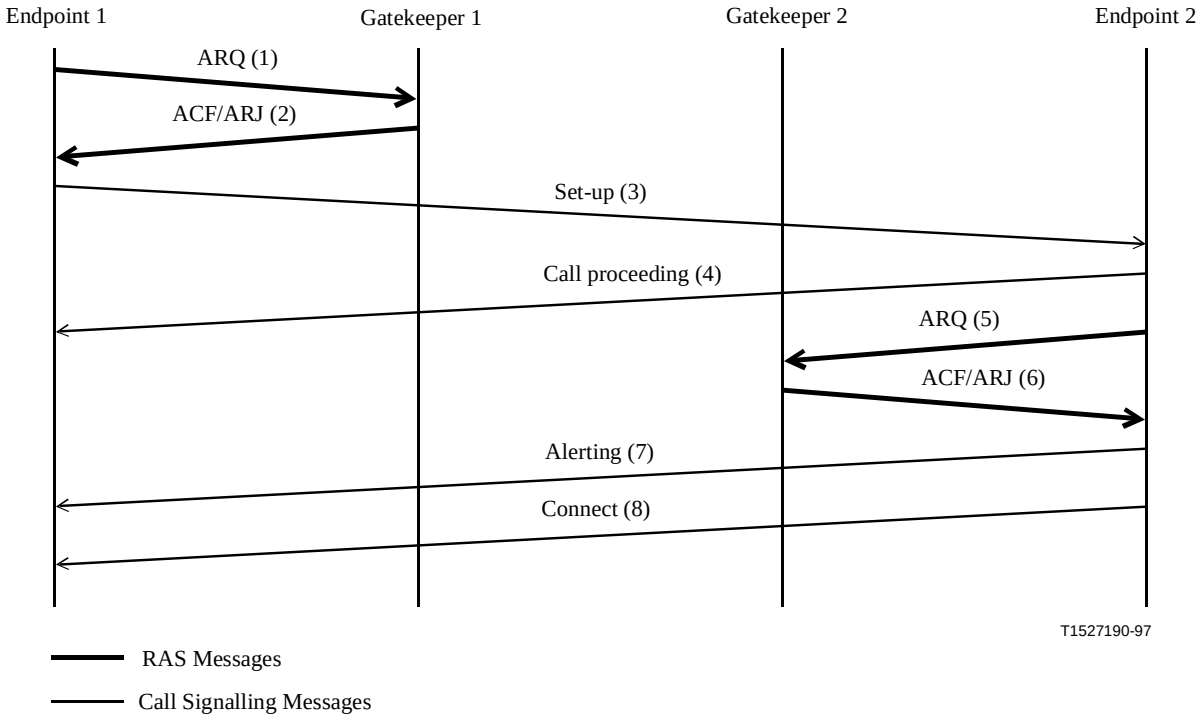


Figure 20/H.323 – Both endpoints registered – Both gatekeepers direct call signalling

In the scenario shown in Figure 21, both endpoints are registered to different Gatekeepers, the calling endpoint's Gatekeeper chooses Direct Call Signalling, and the called endpoint's Gatekeeper chooses to route the call signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with Gatekeeper 1. Gatekeeper 1 may return the Call Signalling Channel Transport Address of endpoint 2 (called endpoint) in the ACF (2) if Gatekeeper 1 has a method of communicating with Gatekeeper 2. Endpoint 1 then sends the Set-up (3) message to either the Transport Address returned by the Gatekeeper (if available) or to the well-known Call Signalling Channel Transport Address of endpoint 2. If endpoint 2 wishes to accept the call, it initiates the ARQ (5)/ACF (6) exchange with Gatekeeper 2. If acceptable, Gatekeeper 2 shall return a Call Signalling Channel Transport Address of itself in the ARJ (6) with a cause code of **routeCallToGatekeeper**. Endpoint 2 replies to endpoint 1 with a Facility (7) message containing the Call Signalling Transport Address of Gatekeeper 2. Endpoint 1 then sends the Release Complete (8) message to endpoint 2. Endpoint 1 shall send a DRQ (9) to Gatekeeper 1 which responds with DCF (10). Endpoint 1 then initiates a new ARQ (11)/ACF (12) exchange with Gatekeeper 1. Endpoint 1 sends a Set-up (13) message to the Gatekeeper's Call Signalling Channel Transport Address. Gatekeeper 2 sends the Set-up (14) message to endpoint 2. Endpoint 2 initiates the ARQ (15)/ACF (16) exchange with Gatekeeper 2. Endpoint 2 then responds with the Connect (18) message which contains its H.245 Control Channel Transport Address for use in H.245 signalling. Gatekeeper 2 sends the Connect (19) message to endpoint 1 which may contain the endpoint 2 H.245 Control Channel Transport Address, or a Gatekeeper 2 (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper chooses to route the H.245 Control Channel or not.

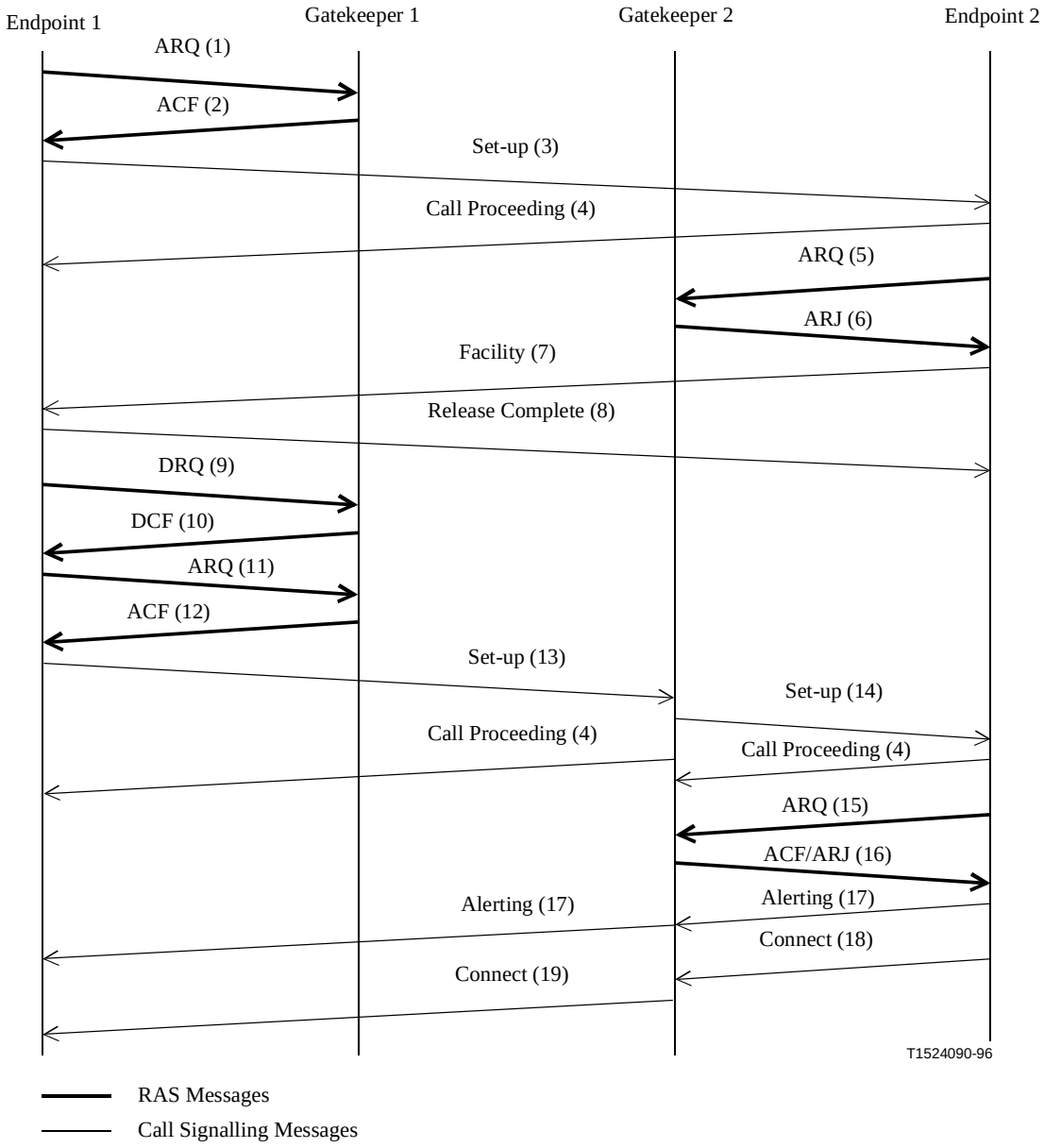


Figure 21/H.323 – Both endpoint registered – Direct/routed call signalling

In the scenario shown in Figure 22, both endpoints are registered to different Gatekeepers, the calling endpoint's Gatekeeper chooses to route the call signalling, and the called endpoint's Gatekeeper chooses Direct Call Signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with Gatekeeper 1. Gatekeeper 1 shall return a Call Signalling Channel Transport Address of itself in the ACF (2). Endpoint 1 then sends the Set-up (3) message using that Transport Address. Gatekeeper 1 then sends the Set-up (4) message containing its Call Signalling Channel Transport Address to the well-known Call Signalling Channel Transport Address of endpoint 2. If endpoint 2 wishes to accept the call, it initiates the ARQ (6)/ACF (7) exchange with Gatekeeper 2. It is possible that an ARJ (7) is received by endpoint 2, in which case it sends Release Complete to endpoint 1. Endpoint 2 responds to Gatekeeper 1 with the Connect (9) message which contains its H.245 Control Channel Transport Address for use in H.245 signalling. Gatekeeper 1 sends the Connect (10) message to endpoint 1 which may contain the endpoint 2 H.245 Control Channel Transport Address, or a Gatekeeper 1 (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper chooses to route the H.245 Control Channel or not.

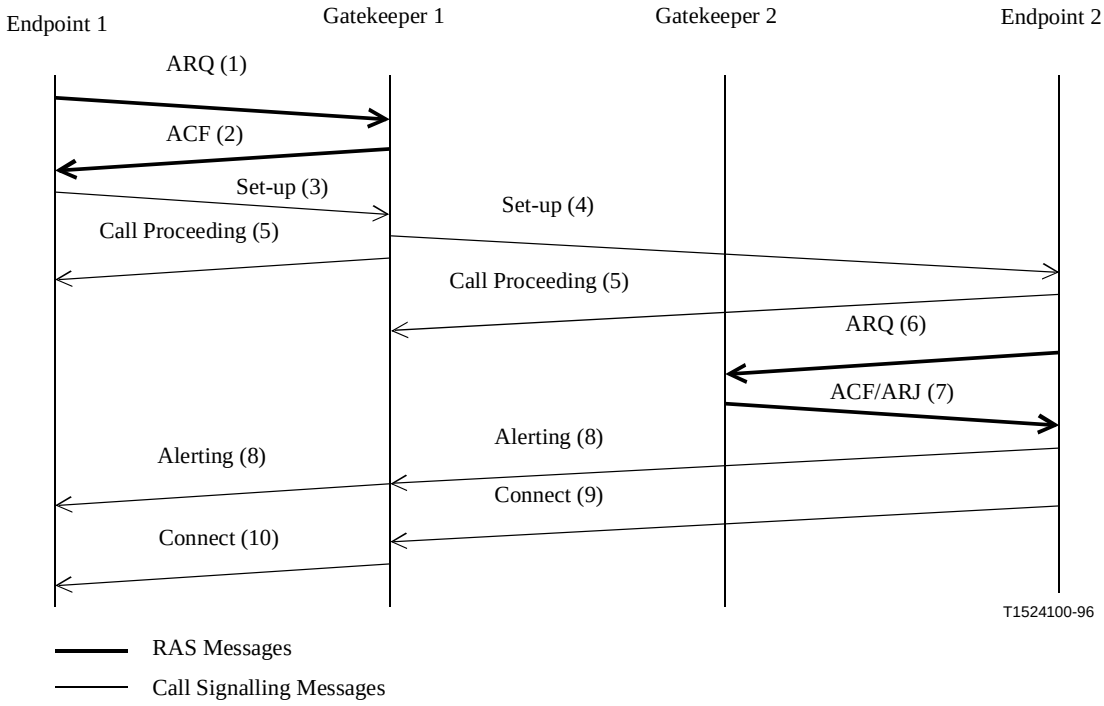


Figure 22/H.323 – Both endpoint registered – Routed/direct call signalling

In the scenario shown in Figure 23, both endpoints are registered to different Gatekeepers, and both Gatekeepers choose to route the call signalling. Endpoint 1 (calling endpoint) initiates the ARQ (1)/ACF (2) exchange with Gatekeeper 1. Gatekeeper 1 shall return a Call Signalling Channel Transport Address of itself in the ACF (2). Endpoint 1 then sends the Set-up (3) message using that Transport Address. Gatekeeper 1 then sends the Set-up (4) message to the well-known Call Signalling Channel Transport Address of endpoint 2. If endpoint 2 wishes to accept the call, it initiates the ARQ (6)/ACF (7) exchange with Gatekeeper 2. If acceptable, Gatekeeper 2 shall return a Call Signalling Channel Transport Address of itself in the ARJ (7) with a cause code of **routeCallToGatekeeper**. Endpoint 2 replies to Gatekeeper 1 with a Facility (8) message containing the Call Signalling Transport Address of Gatekeeper 2. Gatekeeper 1 then sends the Release Complete (9) message to endpoint 2. Gatekeeper 1 sends a Set-up (10) message to Gatekeeper 2's Call Signalling Channel Transport Address. Gatekeeper 2 sends the Set-up (11) message to endpoint 2. Endpoint 2 initiates the ARQ (12)/ACF (13) exchange with Gatekeeper 2. Endpoint 2 then responds to Gatekeeper 2 with the Connect (15) message which contains its H.245 Control Channel Transport Address for use in H.245 signalling. Gatekeeper 2 sends the Connect (16) message to Gatekeeper 1 which may contain the endpoint 2 H.245 Control Channel Transport Address, or a Gatekeeper 2 (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper 2 chooses to route the H.245 Control Channel or not. Gatekeeper 1 sends the Connect (17) message to endpoint 1 which may contain the H.245 Control Channel Transport Address sent by Gatekeeper 2, or a Gatekeeper 1 (MC) H.245 Control Channel Transport Address, based on whether the Gatekeeper 1 chooses to route the H.245 Control Channel or not.

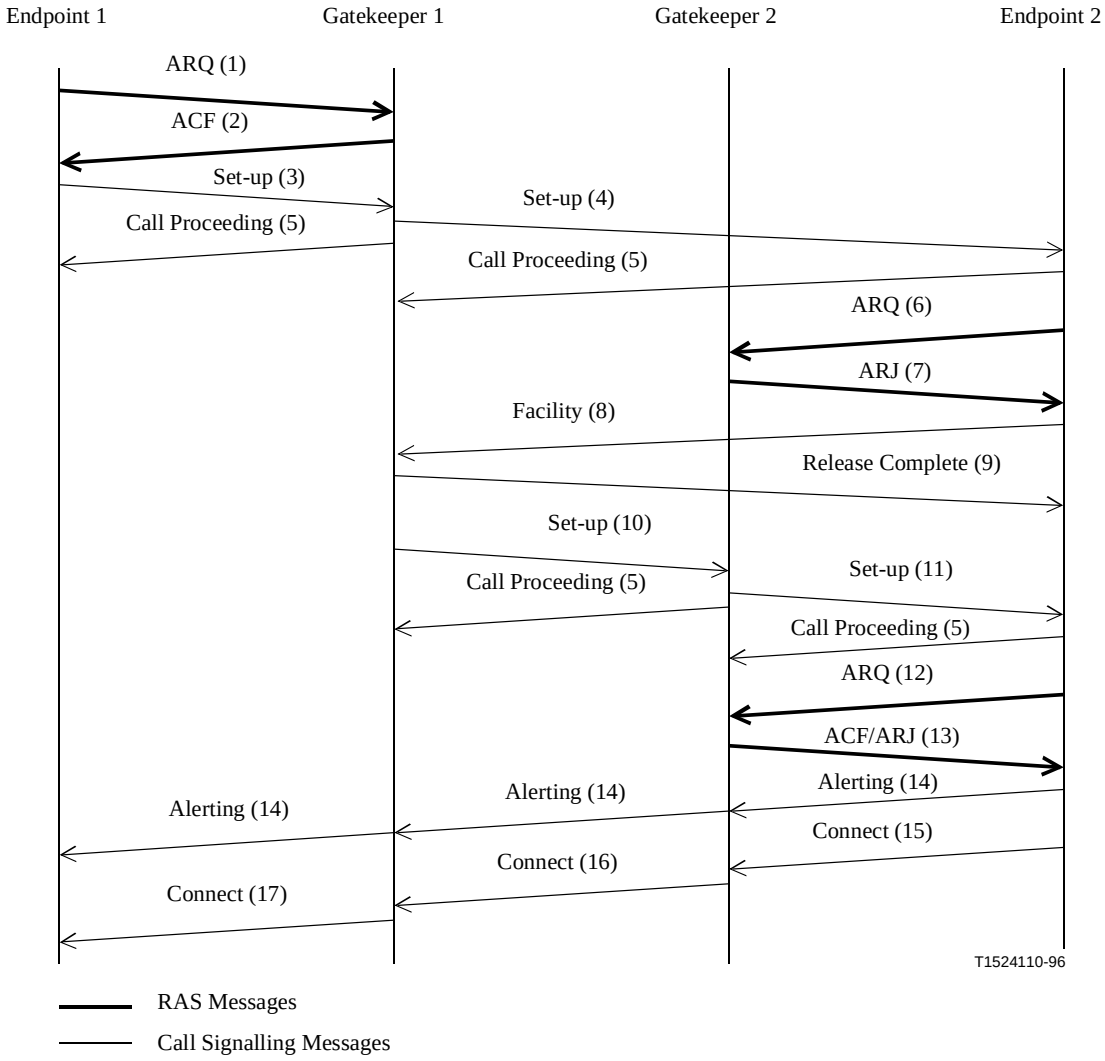


Figure 23/H.323 – Both endpoint registered – Both Gatekeepers routing call signalling

8.1.6 Call set-up via gateways

8.1.6.1 Gateway inbound call set-up

When an external terminal calls a LAN endpoint via the Gateway, call set-up between the Gateway and the LAN endpoint proceeds the same as the endpoint-to-endpoint call set-up. The Gateway may need to issue Call Proceeding messages to the external terminal while establishing the call on the network.

A Gateway which cannot directly route an incoming SCN call to an H.323 endpoint shall be able to accept two-stage dialling. For Gateways to H.320 networks (also H.321, H.322 and H.310 in H.321 mode), the Gateway shall accept SBE numbers from the H.320 terminal. For Gateways to H.310 native mode and H.324 networks, the Gateway shall accept H.245 **userInputIndication** messages from the H.324 terminal. In these two cases, support of DTMF is optional. For Gateways to speech only terminals, the Gateway shall accept DTMF numbers from the speech only terminal. These numbers will indicate a second stage dialling number to access the individual endpoint on the LAN.

8.1.6.2 Gateway outbound call set-up

When a LAN endpoint calls an external terminal via the Gateway, call set-up between the LAN endpoint and the Gateway proceeds the same as the endpoint-to-endpoint call set-up. The Gateway will receive the destination E.164 address in the Set-up message. It will then use this address to place the outbound call. The Gateway may issue Call Proceeding messages to the LAN endpoint while establishing the outgoing call.

The Progress Indicator information element is used to indicate that inter-networking is occurring. The Gateway shall issue a Progress indicator information element within the Alerting, Call Proceeding or Connect messages. This information may also be sent in a Progress message.

The LAN endpoint shall send all E.164 addresses that it is calling in the Set-up message. For example, a six B-channel call on the ISDN will require six E.164 addresses in the Set-up message. The Gateway shall respond to the Set-up message with a Connect or Release Complete message as well as Alerting, Call Proceeding, or Progress messages. Failure of the SCN call shall be reported to the LAN endpoint in the Release Complete message. The use of multiple CRV values and multiple Set-up messages is for further study. Addition of channels on the SCN during a call is for further study.

A LAN endpoint which is registered with a Gatekeeper should request sufficient call bandwidth in the ARQ message for the aggregate of all SCN calls. If sufficient call bandwidth was not requested in the ARQ message, the procedures of 8.4.1, Bandwidth Changes, shall be followed in order to obtain additional call bandwidth.

The Gateway may advance to Phase B after placing the first call on the SCN. Additional calls for the additional SCN E.164 numbers may be placed after the capability exchange with the Gateway and establishment of audio communications with the SCN endpoint.

8.1.7 Call set-up with an MCU

For Centralized Multipoint Conferences, all endpoints exchange call signalling with the MCU. Call set-up between an endpoint and the MCU proceeds the same as the endpoint-to-endpoint call set-up scenarios of 8.1.1 through 8.1.5. The MCU may be the called endpoint or the calling endpoint.

In a Centralized Multipoint Conference, the H.245 Control Channel is opened between the endpoints and the MC within the MCU. The audio, video, and data channels are opened between the endpoints and the MP within the MCU. In a Decentralized Multipoint Conference, the H.245 Control Channel is open between the endpoint and the MC (there may be many such H.245 Control Channels, one for each call). The Audio and Video Channels should be multicast to all endpoints in the conference. The Data Channel shall be opened with the Data MP.

In an ad hoc Multipoint Conference where there is no MC within the endpoints, the H.245 Control Channel shall be routed through the Gatekeeper. Initially, the H.245 Control Channel will be routed between the endpoints through the Gatekeeper. When the conference switches to multipoint, the Gatekeeper may connect the endpoints to an MC associated with the Gatekeeper.

In an ad hoc Multipoint Conference where one or both of the endpoints contains an MC, the normal call set-up procedures defined in 8.1.1 through 8.1.5 are used. The master-slave determination procedure is used to determine which MC will be the active MC for the conference.

8.1.8 Call forwarding

An endpoint wishing to forward a call to another endpoint may issue a Facility message indicating the address of the new endpoint. The endpoint receiving this Facility indication should send a Release Complete and then restart the Phase A procedures with the new endpoint.

8.1.9 Broadcast call set-up

This subclause is for further study.

8.2 Phase B – Initial communication and capability exchange

Once both sides have exchanged call set-up messages from Phase A, the endpoints shall establish the H.245 Control Channel. The procedures of Recommendation H.245 are used over the H.245 Control Channel for the capability exchange and to open the media channels.

NOTE – Optionally, the H.245 Control Channel may be set up by the called endpoint on receipt of Set-up, and by calling endpoint on receipt of Alerting or Call Proceeding. In the event that Connect does not arrive, or an endpoint sends Release Complete, the H.245 Control Channel shall be closed.

Endpoint system capabilities are exchanged by transmission of the H.245 **terminalCapabilitySet** message. This capability message shall be the first H.245 message sent.

The master-slave determination procedure of H.245 shall take place as described in 6.2.8.4. In cases where both endpoints in a call have MC capability, the master-slave determination is used for determining which MC will be the active MC for the conference. The active MC may then send the **mcLocationIndication** message. The procedure also provides master-slave determination for opening bidirectional channels for data.

If the initial capability exchange or master-slave determination procedures fail, these should be retried at least two additional times before the terminal abandons the connection attempt and proceeds to Phase E.

Following this exchange of capabilities, the endpoints shall proceed directly to the desired operating mode, i.e. Phase C.

8.3 Phase C – Establishment of audiovisual communication

Following the exchange of capabilities and master-slave determination, the procedures of Recommendation H.245 shall then be used to open logical channels for the various information streams. The audio and video streams, which are transmitted in the logical channels set-up in H.245, are transported over dynamic TSAP Identifiers using an unreliable protocol (see Recommendation H.225.0). Data communications which is transmitted in the logical channels set-up in H.245, are transported using a reliable protocol (see Recommendation H.225.0).

The **openLogicalChannelAck** returns the Transport Address that the receiving endpoint has assigned to that logical channel. The transmitting channel shall then send the information stream associated with the logical channel to that Transport Address.

Following the opening of logical channels for audio and video, one **h2250MaximumSkewIndication** message shall be sent by the transmitter for each associated audio and video pair.

8.3.1 Mode changes

During a session, the procedures for changing channel structure, capability, receive mode etc. shall be carried out as defined in Recommendation H.245.

8.3.2 Exchange of video by mutual agreement

The indication **videoIndicateReadyToActivate** is defined in Recommendation H.245. Its use is optional, but when used the procedure shall be as follows.

Endpoint 1 has been set so that video is not transmitted unless, and until, endpoint 2 has also indicated readiness to transmit video. Endpoint 1 shall send the indication

videoIndicateReadyToActivate when the initial capability exchange has been completed, but shall not transmit a video signal until it has received either **videoIndicateReadyToActivate** or incoming video from endpoint 2.

An endpoint which has not been set in this optional way is not obliged to wait until receipt of **videoIndicateReadyToActivate** or video before initiating its video transmission.

8.3.3 Media Stream Address Distribution

In unicast, the endpoint shall open logical channels to the MCU or other endpoint. Addresses are passed in the **openLogicalChannel** and **openLogicalChannelAck**.

In multicast, the multicast addresses are assigned by the MC and distributed to the endpoints in the **communicationsModeCommand**. It is the responsibility of the MC to allocate and assign unique multicast addresses. The endpoint shall signal an open logical channel to the MC with a multicast address in the **openLogicalChannel**. The MC shall forward the **openLogicalChannel** to each receiving endpoint.

In multi-unicast, the endpoint must open logical channels to each of the other endpoints. The **openLogicalChannel** is sent to the MC and shall contain the terminal number of the endpoint for which the channel is intended. The endpoint can match a **openLogicalChannelAck** by the **forwardLogicalChannelNumber**.

8.4 Phase D – Call services

8.4.1 Bandwidth changes

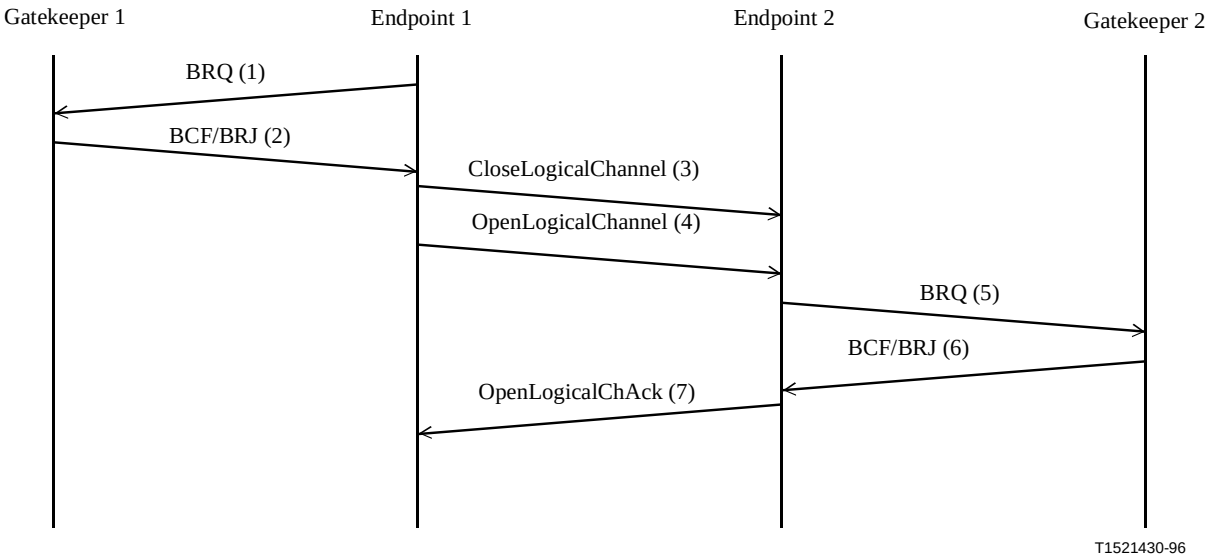
Call bandwidth is initially established and approved by the Gatekeeper during the admissions exchange. An endpoint shall assure that the aggregate for all transmitted and received audio and video channels, excluding any RTP headers, RTP payload headers, LAN headers, and other overhead, is within this bandwidth. Data and control channels are not included in this limit.

At any time during a conference, the endpoints or Gatekeeper may request an increase or decrease in the call bandwidth. An endpoint may change the bit rate of a logical channel without requesting a bandwidth change from the Gatekeeper if the aggregate bit rate of all transmitted and received channels does not exceed the current call bandwidth. If the change will result in a aggregate bit rate that exceeds the current call bandwidth, the endpoint shall request a change in the call bandwidth from its Gatekeeper and await confirmation prior to actually increasing any bit rate. A bandwidth change request is recommended when an endpoint will use a reduced bandwidth for an extended period of time, thus freeing up bandwidth for other calls.

An endpoint wishing to change its call bandwidth sends a Bandwidth Change Request (BRQ) message (1) to the Gatekeeper. The Gatekeeper determines if the request is acceptable. The criteria for this determination is outside the scope of this Recommendation. If the Gatekeeper determines that the request is not acceptable, it returns a Bandwidth Change Reject (BRJ) message (2) to endpoint. If the Gatekeeper determines that the request is acceptable, it returns a Bandwidth Change Confirm (BCF) message (2).

If endpoint 1 wishes to increase its transmitted bit rate on a logical channel, it first determines if the call bandwidth will be exceeded. See Figure 24. If it will, endpoint 1 shall request a bandwidth change (1 and 2) from Gatekeeper 1. When the call bandwidth is sufficient to support the change, endpoint 1 sends a **closeLogicalChannel** (3) message to close the logical channel. It then reopens the logical channel using the **openLogicalChannel** (4) specifying the new bit rate. If the receiving endpoint wishes to accept the channel with the new bit rate, it must first assure that its call bandwidth is not exceeded by the change. If it is, the endpoint shall request a call bandwidth change (5 and 6) with its Gatekeeper. When the call bandwidth is sufficient to support the channel, the

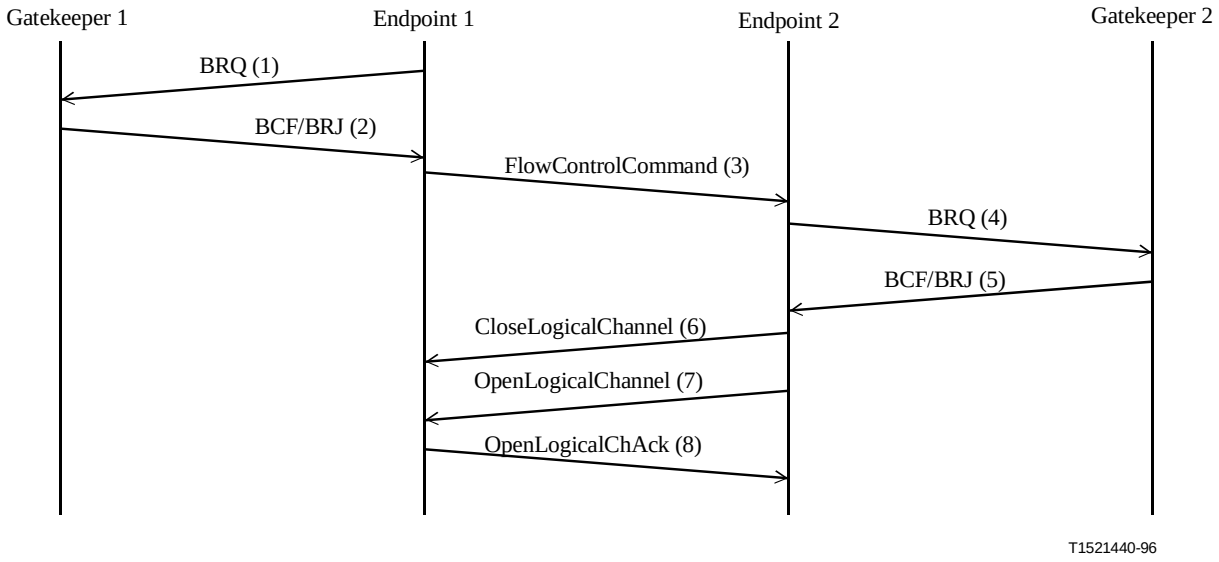
endpoint replies with an **openLogicalChannelAck** (7), otherwise, it responds with an **openLogicalChannelReject** indicating unacceptable bit rate.



NOTE – Gatekeeper 1 and Gatekeeper 2 may be the same Gatekeeper.

Figure 24/H.323 – Bandwidth change request – Transmitter change

If the endpoint 1 wishes to increase its transmitted bit rate on a logical channel from endpoint 2, which it previously flow controlled to a lower bit rate, endpoint 1 first determines if the call bandwidth will be exceeded. See Figure 25. If it will, endpoint 1 shall request a bandwidth change from Gatekeeper 1. When the call bandwidth is sufficient to support the change, endpoint 1 sends a **flowControlCommand** (3) to indicate the new upper limit on bit rate for the channel. If endpoint 2 decides to increase the bit rate on the channel, it must first assure that its call bandwidth is not exceeded by the change. If it is, endpoint 2 shall request a call bandwidth change (4 and 5) with its Gatekeeper. When the call bandwidth is sufficient to support the channel, endpoint 2 will send the **closeLogicalChannel** (6) message to close the logical channel. It then reopens the logical channel using the **openLogicalChannel** (7) specifying the new bit rate. Endpoint 1 should then accept the channel with the new bit rate, and it replies with an **openLogicalChannelAck** (6).



NOTE – Gatekeeper 1 and Gatekeeper 2 may be the same Gatekeeper.

Figure 25/H.323 – Bandwidth change request – Receiver change

A Gatekeeper wishing to change the transmitted bit rate of endpoint 1 sends a BRQ message to endpoint 1. If the request is for a decrease in bit rate, endpoint 1 shall always comply by reducing its aggregate bit rate and returning a BCF. Endpoint 1 may initiate the appropriate H.245 signalling to inform endpoint 2 that bit rates have changed. This will allow endpoint 2 to inform its Gatekeeper of the change. If the request is for an increase, the endpoint may increase its bit rate when desired.

8.4.2 Status

In order for the Gatekeeper to determine if an endpoint is turned off, or has otherwise entered a failure mode, the Gatekeeper may use the Information Request (IRQ)/Information Request Response (IRR) message sequence (see Recommendation H.225.0) to poll the endpoints at an interval decided by the manufacturer. The polling interval shall be greater than 10 s. This message may also be used by a diagnostic device as described in 11.2.

The Gatekeeper may want an endpoint to periodically send an unrequested IRR message. The Gatekeeper may indicate this to the endpoint by specifying the rate that this IRR is sent within the irrFrequency field of the Admission Confirm (ACF) message. An endpoint receiving this irrFrequency rate shall send an IRR message at that rate for the duration of the call. While this rate is in effect, the Gatekeeper may still send IRQ messages to the endpoint which shall respond as described above.

During the duration of a call, an endpoint or Gatekeeper may periodically request call status from another endpoint. The requesting endpoint or Gatekeeper issues a Status Enquiry message. The Endpoint receiving the Status Enquiry message shall respond with a Status message indicating the current call state. This procedure may be used by the Gatekeeper in order to periodically check if a call is still active. Note that this is a H.225.0 message sent on the Call Signalling Channel, and should not be confused with IRR which is a RAS message sent on the RAS Channel.

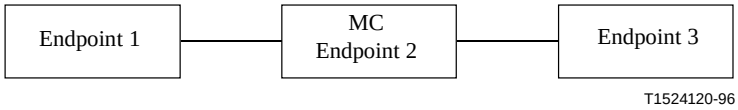
8.4.3 Ad Hoc Conference Expansion

The following procedures are optional for terminals and Gateways and mandatory for MCs.

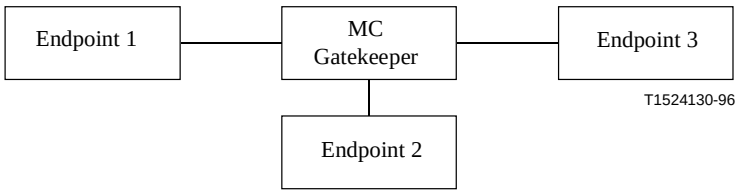
An Ad Hoc Multipoint conference is one that can be expanded from a point-to-point conference involving an MC to a multipoint conference. First, a point-to-point conference is created between

two endpoints (endpoint 1 and endpoint 2). At least one endpoint, or the Gatekeeper, must contain an MC. Once the point-to-point conference has been created, the conference may be expanded to multipoint conference in two different ways. The first way is when any endpoint in the conference invites another endpoint (endpoint 3) into the conference by calling that endpoint through the MC. The second way is for an endpoint (endpoint 3) to join an existing conference by calling an endpoint in the conference.

Ad Hoc Conference expansion can take place when using either the Direct Call Signalling model or the Gatekeeper Routed Call Signalling model. The H.245 Control Channel topology for the Direct Call Signalling model appears as:



The H.245 Control Channel topology for the Gatekeeper Routed Call Signalling model appears as:



In either case an MC must be present in the conference at the time of expansion to any number greater than 2 endpoints.

The procedures required to create a point-to-point conference and then expand the conference through invite and join, for each call model, is covered in the following subclauses.

It should be noted that the call is ended by a failure of the entity that is providing the MC.

8.4.3.1 Direct Endpoint Call Signalling – Conference Create

Endpoint 1 creates a conference with endpoint 2 as follows:

A1) Endpoint 1 sends a Set-up message to endpoint 2 containing a globally unique CID = N and conferenceGoal = create according to the procedure in 8.1.

A2) Endpoint 2 has the following options:

A2a) If it wants to join the conference, it sends a Connect message with CID = N to endpoint 1. In this case it is either:

1) not participating in another conference; or

2) it is participating in another conference, it is capable of participating in multiple conferences at the same time, and the received CID = N does not match the CID of any of the conferences in which it is currently participating.

A2b) If it is in another conference with CID = M and can participate in only one conference at a time it either:

1) rejects the call by sending Release Complete indicating in-conference; or

2) it can request endpoint 1 to join the conference with CID = M by sending a Facility message indicating **routeCallToMC** with the Call Signalling Channel Transport Address of the endpoint containing the MC and CID = M of the conference.

A2c) If it does not wish to join this conference, it rejects the call by sending Release Complete indicating destinationBusy.

A3) If endpoint 2 enters the conference, endpoint 1 uses the transport address of the Control Channel provided in the Connect message to open the Control Channel with endpoint 2.

A4) The H.245 messages are then exchanged as described below:

A4a) **TerminalCapabilitySet** messages are exchanged between the endpoints to determine the version number of the H.245 used in order to parse the remaining received messages correctly.

A4b) Using H.245 master-slave determination procedure, it is determined that endpoint 2 is the master. In the Gatekeeper-Routed model, the master could be in the Gatekeeper's MC. If the master has an MC, it becomes the Active MC. It may then send the **MCLocationIndication** to the other endpoint(s). The MC may be active in the conference now, or when the user initiates the multipoint conference function, at the choice of the manufacturer.

A4c) The master may send the **terminalNumberAssign** message to the endpoints. The endpoints shall use the 8-bit terminal number, and not use the 8-bit MCU number, from the 16-bit number assigned as the low 8 bits of the SSRC field in the RTP header. These low 8 bits in SSRC then identify the streams from a particular endpoint.

A4d) Since the capabilities of the receiver are known from the **TerminalCapabilitySet** message, the transmitter opens the logical channels. It shall send one **h2250MaximumSkewIndication** for each pair of audio and video transmitted.

8.4.3.2 Direct Endpoint Call Signalling – Conference Invite

There are two cases of the conference invite. First, the endpoint which contains the active MC wishes to invite another endpoint into the conference. Second, an endpoint which does not contain the active MC wishes to invite another endpoint into the conference.

After a point-to-point conference has been established using procedures A1 to A4, an endpoint (endpoint 2) containing the active MC wishing to add another endpoint to the conference shall use the following procedure:

B1) Endpoint 2 sends a Set-up message to endpoint 3 with CID = N and conferenceGoal = invite according to the procedures in 8.1. See Figure 26.

B2) Endpoint 3 has the following options:

B2a) If it wishes to accept the invitation to join the conference, it sends a Connect message with CID = N to endpoint 2.

B2b) If it wishes to reject the invitation to join the conference, it sends a Release Complete message indicating destinationBusy to endpoint 2.

B2c) If it is in another conference with CID = M, it can request endpoint 2 to join the conference with CID = M by sending a Facility message indicating **routeCallToMC** with the Call Signalling Channel Transport Address of the endpoint containing the MC and CID = M of the conference.

B2d) If the received CID matches the CID of a conference that endpoint 3 is currently participating in, it shall reject the call by sending Release Complete indicating that it is already in the conference.

B3) If endpoint 3 accepts the invitation, endpoint 2 uses the transport address of the Control Channel provided in the Connect message to open the Control Channel with endpoint 3.

B4) The H.245 messages are then exchanged as described below:

C1) **TerminalCapabilitySet** messages are exchanged between the MC and endpoint 3.

C2) Using H.245 master-slave determination procedure, it is determined that endpoint 2 is already the active MC. The MC may then send the **MCLocationIndication** to the endpoint 3.

C3) The MC shall send **multipointModeCommand** at this time to all the three endpoints.

C4) The MC may send the **terminalNumberAssign** message to endpoint 3. If received, the endpoints shall use the 8-bit terminal number, and not use the 8-bit MCU number, from the 16-bit number assigned as the low 8 bits of the SSRC field in the RTP header. These low 8 bits in SSRC then identify the streams from a particular endpoints.

C5) An endpoint can get the list of the other endpoints in the conference by sending the **terminalListRequest** message to the MC. The MC responds with the **terminalListResponse**.

C6) Whenever a new endpoint joins the conference, the MC sends the **terminalNumberAssign** message to endpoint 4 and **terminalJoinedConference** message to endpoints 1, 2 and 3.

C7) Whenever an endpoint leaves the conference, the MC sends **terminalLeftConference** to the remaining endpoints.

C8) The MC shall send the **communicationModeCommand** to all the endpoints in the conference.

C9) Endpoint 1 and endpoint 2 will close their logical channels that were created during the point-to-point conference if they are inconsistent with the information contained in the **communicationModeCommand**.

C10) The logical channels can now be opened between the MC and the endpoints.

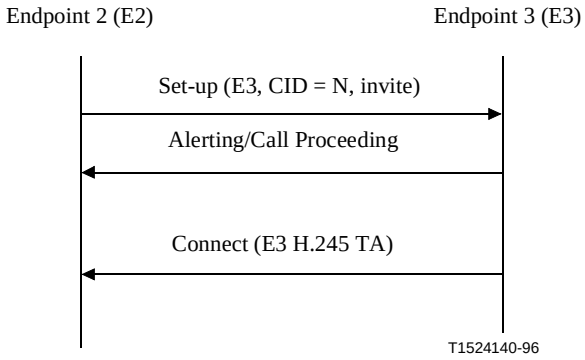


Figure 26/H.323 – MC invite signalling

After a point-to-point conference has been established using procedures A1 to A4, an endpoint (endpoint 1) that does not contain the active MC wishing to add another endpoint to the conference, shall use the following procedure:

B1) Endpoint 1 sends a Set-up message to the MC (endpoint 2) with a new CRV indicating a call to endpoint 3 by providing the transport address of endpoint 3, CID = N, and conferenceGoal = invite. See Figure 27.

B2) Endpoint 2 sends a Set-up message to endpoint 3 with CID = N and conferenceGoal = invite according to the procedures in 8.1.

B3) During call signalling with endpoint 3, endpoint 2 shall pass Call Signalling messages received from endpoint 3, including Connect, to endpoint 1 (the original inviter).

B4) Endpoint 3 has the same options, described previously, of either accepting or rejecting the invitation.

B5) At some time after the completion of the call set-up procedure between endpoint 2 and endpoint 3, endpoint 2 shall send a Release Complete message to endpoint 1.

B6) If endpoint 3 accepts the invitation, endpoint 2 uses the transport address of the Control Channel provided in the Connect message to open the Control Channel with endpoint 3.

B7) The H.245 messages are then exchanged as previously described in procedures C1 to C10.

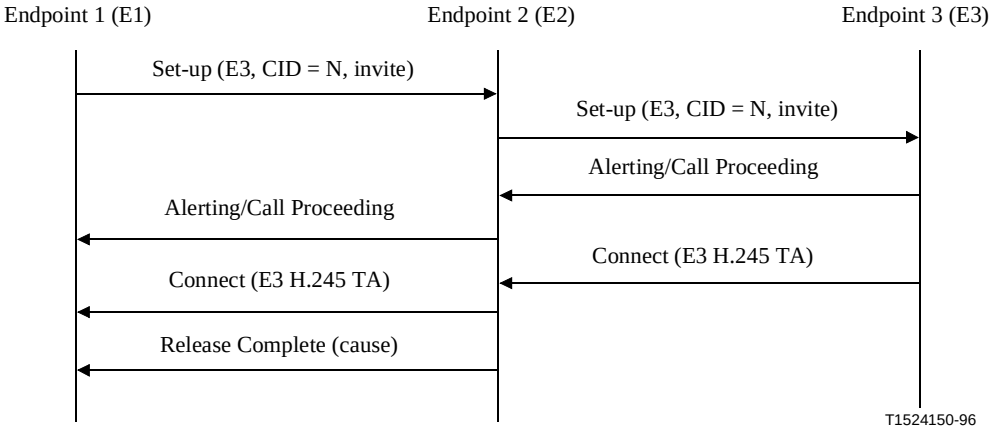


Figure 27/H.323 – Non-MC invite signalling

8.4.3.3 Direct Endpoint Call Signalling – Conference Join

There are two cases of the conference join. First, an endpoint calls the endpoint which contains the active MC. Second, an endpoint calls an endpoint which is not the active MC.

After a point-to-point conference has been established using procedures A1 to A4, an endpoint (endpoint 3) wishing to join a conference may attempt to connect with the endpoint containing the active MC in the conference. In this case, the following procedure shall be used:

B1) Endpoint 3 sends a Set-up message to endpoint 2 with CID = N, and conferenceGoal = join according to the procedure in 8.1. See Figure 28.

B2) If the CID matches the CID of an active conference in the MC, endpoint 2 (MC) has the following options:

- B2a) If it decides that endpoint 3 should be allowed to join the conference, it sends the Connect message with CID = N.
- B2b) If it decides that endpoint 3 should not be allowed to join the conference, it sends the Release Complete message with destinationBusy.

B3) If the CID does not match the CID of an active conference in the MC, endpoint 2 shall send Release Complete indicating a bad CID.

B4) If endpoint 2 allows the join, endpoint 2 opens the Control Channel with endpoint 3.

B5) The H.245 messages are then exchanged as previously described in procedures C1 to C10.

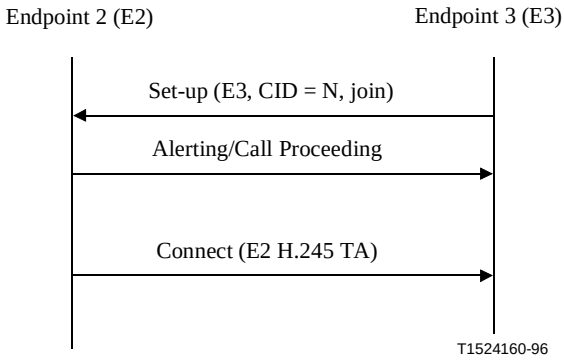


Figure 28/H.323 – MC join signalling

After a point-to-point conference has been established using procedures A1 to A4, an endpoint (endpoint 3) wishing to join a conference may attempt to connect with an endpoint that does not contain the active MC in the conference. In this case, the following procedure shall be used:

B1) Endpoint 3 sends a Set-up message to endpoint 1 with CID = N, and conferenceGoal = join according to the procedure in 8.1. See Figure 29.

B2) Endpoint 1 returns a Facility message indicating **routeCallToMC** with the Call Signalling Channel Transport Address of endpoint 2 (containing the active MC) and the CID = N of the conference.

B3) Endpoint 3 then sends a Set-up message to endpoint 2 (MC) with CID = N and conferenceGoal = join as described in the previous conference join procedure.

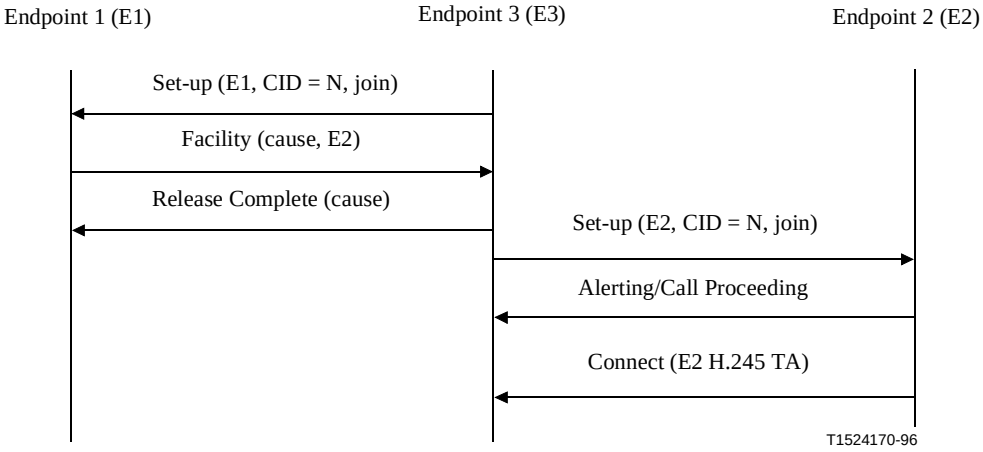


Figure 29/H.323 – Non-MC join signalling

8.4.3.4 Gatekeeper Routed Call Signalling – Conference Create

In cases where the Gatekeeper routes the Call Signalling Channel and the H.245 Control Channel, the Gatekeeper should contain (or have access to) an MC or MCU. Procedures A1 to A4 are used to establish the point-to-point call. During master-slave determination (A4b), if the Gatekeeper's terminalType is greater than the **terminalType** received from an endpoint in the **masterSlaveDetermination** message, the Gatekeeper may replace the endpoint's **terminalType** value with its own before forwarding the message to the destination endpoint. If the Gatekeepers **terminalType** is not greater than the **terminalType** of the endpoint, the Gatekeeper shall not modify the **terminalType** value. In effect, the Gatekeeper is performing the master-slave determination procedure with each endpoint. If the Gatekeeper wins the master-slave determination with both endpoints, the MC associated with the Gatekeeper shall be the active MC, otherwise, one of the endpoints shall be the active MC.

8.4.3.5 Gatekeeper Routed Call Signalling – Conference Invite

After a point-to-point conference has been established using procedures A1 to A4 as modified above, an endpoint (endpoint 1 or 2) that does not contain the active MC wishing to add another endpoint to the conference, shall use the following procedure:

B1) Endpoint 1 sends a Set-up message through the Gatekeeper directed to endpoint 3 with a new CRV, CID = N, and conferenceGoal = invite. See Figure 30.

B2) The Gatekeeper (MC) sends a Set-up message to endpoint 3 with CID = N and conferenceGoal = invite according to the procedures in 8.1.

B3) During call signalling with endpoint 3, the Gatekeeper shall pass Call Signalling messages received from endpoint 3, including Connect, to endpoint 1 (the original inviter).

B4) Endpoint 3 has the same options, described previously, of either accepting or rejecting the invitation.

B5) At some time after the completion of the call set-up procedure between the Gatekeeper and endpoint 3, the Gatekeeper shall send a Release Complete message to endpoint 1.

B6) If endpoint 3 accepts the invitation, the Gatekeeper uses the transport address of the Control Channel provided in the Connect message to open the Control Channel with endpoint 3.

B7) The H.245 messages are then exchanged as previously described in procedures C1 to C10 with the Gatekeeper taking part in all master-slave determination procedures as the active MC (C2). At this time, the Control Channels from the endpoints should be connected to the MC, and the MC should be in control of the conference.

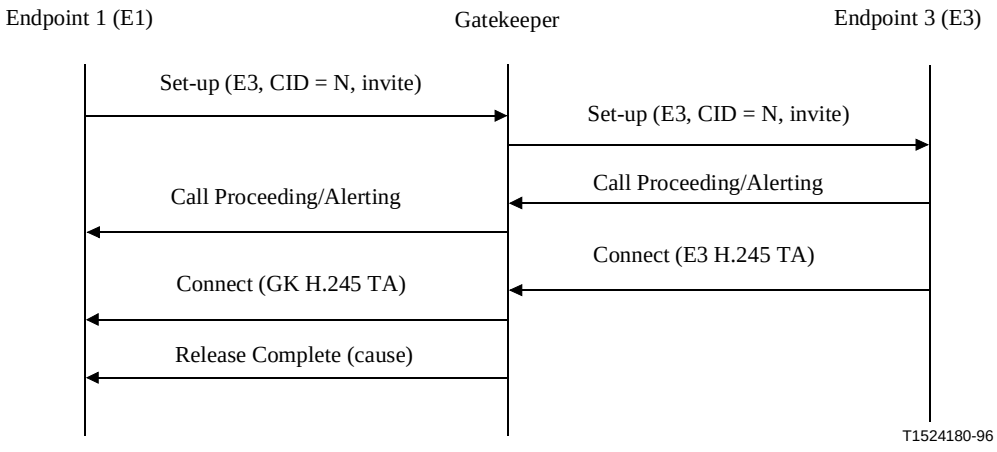


Figure 30/H.323 – Gatekeeper routed invite signalling

8.4.3.6 Gatekeeper Routed Call Model – Conference Join

After a point-to-point conference has been established using procedures A1 to A4 as modified above, an endpoint (endpoint 3), wishing to join a conference may attempt to connect with an endpoint that does not contain the active MC in the conference. In this case, the following procedure shall be used:

B1) Endpoint 3 sends a Set-up message through the Gatekeeper directed to endpoint 1 with CID = N, and conferenceGoal = join according to the procedure in 8.1. See Figure 31.

B2) If the CID matches the CID of an active conference in the MC, the Gatekeeper (MC) has the following options:

- B2a) If it decides that endpoint 3 should be allowed to join the conference, it sends the Connect message with CID = N to endpoint 3.
- B2b) If it decides that endpoint 3 should not be allowed to join the conference, it sends the Release Complete message with destinationBusy.
- B2c) The Gatekeeper may forward the Set-up message to endpoint 1. Endpoint 1 may respond with a Facility message indicating **routeCallToMC** or it may respond with a release complete.

B3) If the CID does not match the CID of an active conference in the MC, the Gatekeeper shall send Release Complete indicating a bad CID.

B4) If the Gatekeeper allows the join, the Gatekeeper uses the transport address of the Control Channel provided in the Set-up message to open the Control Channel with endpoint 3.

B5) The H.245 messages are then exchanged as previously described in procedures C1 to C10 with the Gatekeeper taking part in all master-slave determination procedures as the active MC (C2). At this time, the Control Channels from the endpoints should be connected to the MC, and the MC should be in control of the conference.

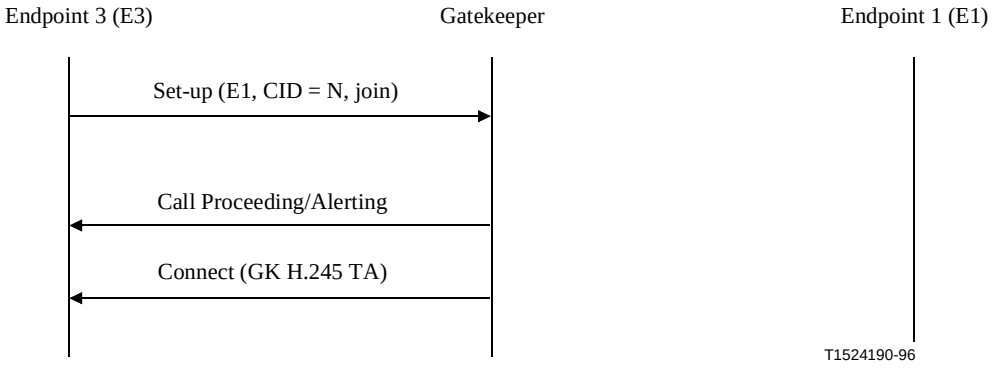


Figure 31/H.323 – Gatekeeper routed join signalling

8.4.4 Supplementary services

Provisions have been made within this Recommendation to allow endpoints to the use the Supplementary Services as defined in Q.931-, Q.932- and the Q.95X-Series of Recommendations. These services shall use the Call Signalling Channel and Q.931 messages. These services may provide features such as hold, transfer, forward and redirection.

Other methods of providing supplementary services are for further study.

8.5 Phase E – Call termination

Either endpoint may terminate a call by the following procedure:

1) It should discontinue transmission of video at the end of a complete picture, and then close all logical channels for video.
2) It should discontinue transmission of data and then close all logical channels for data.
3) It should discontinue transmission of audio and then close all logical channels for audio.
4) It shall transmit the H.245 **endSessionCommand** message in the H.245 Control Channel, indicating to the far end that it wishes to disconnect the call and then discontinue H.245 message transmission. It shall then close the H.245 Control Channel.
5) It shall then wait to receive the **endSessionCommand** message from the other endpoint and then shall close the H.245 Control Channel.
6) If the Call Signalling Channel is open, a Release Complete message shall be sent and the channel closed.
7) It shall clear the call by using the procedures defined below.

An endpoint receiving **endSessionCommand** without first having transmitted it, shall carry out steps 1) to 7) above, except that in step 5), it shall not wait for the **endSessionCommand** from the first endpoint.

Terminating a call may not terminate a conference; a conference may be explicitly terminated using an H.245 message (**dropConference**). In this case, the endpoints shall wait for the MC to terminate the calls as described above.

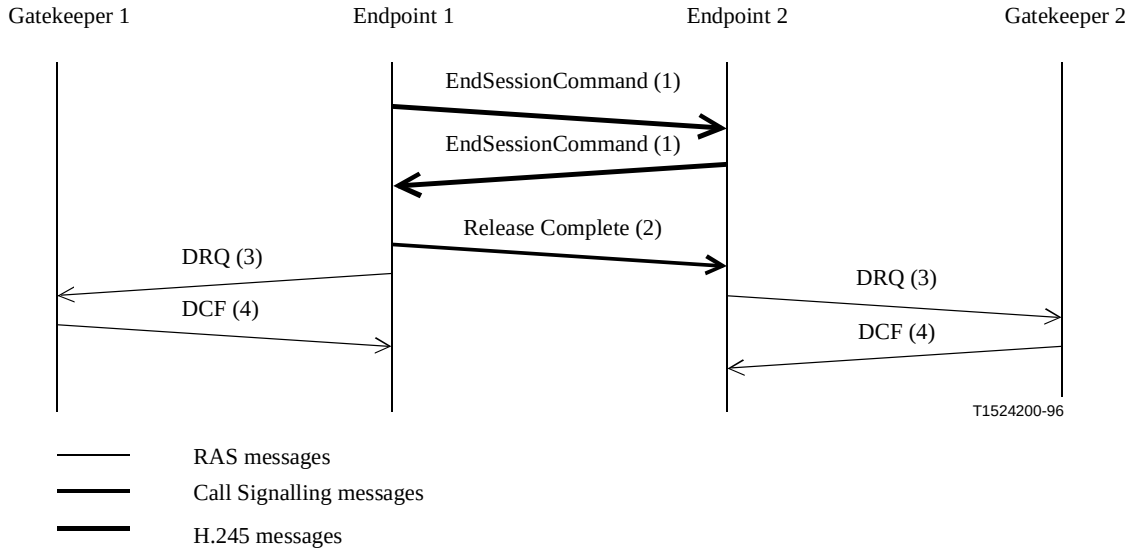
8.5.1 Call clearing without a Gatekeeper

In networks that do not contain a Gatekeeper, after steps 1) to 6) above, the call is terminated. No further action is required.

8.5.2 Call clearing with a Gatekeeper

In networks that contain a Gatekeeper, the Gatekeeper needs to know about the release of bandwidth. After performing steps 1) to 6) above, each endpoint shall transmit an H.225.0 Disengage Request (DRQ) message 3) to its Gatekeeper. The Gatekeeper shall respond with a Disengage Confirm (DCF) message 4). After sending the DRQ message, the endpoints shall not send further unsolicited IRR messages to the Gatekeeper. See Figure 32. At this point, the call is terminated. Figure 32 shows the direct call model; a similar procedure is followed for the Gatekeeper routed model.

The DRQ and DCF messages shall be sent on the RAS Channel.



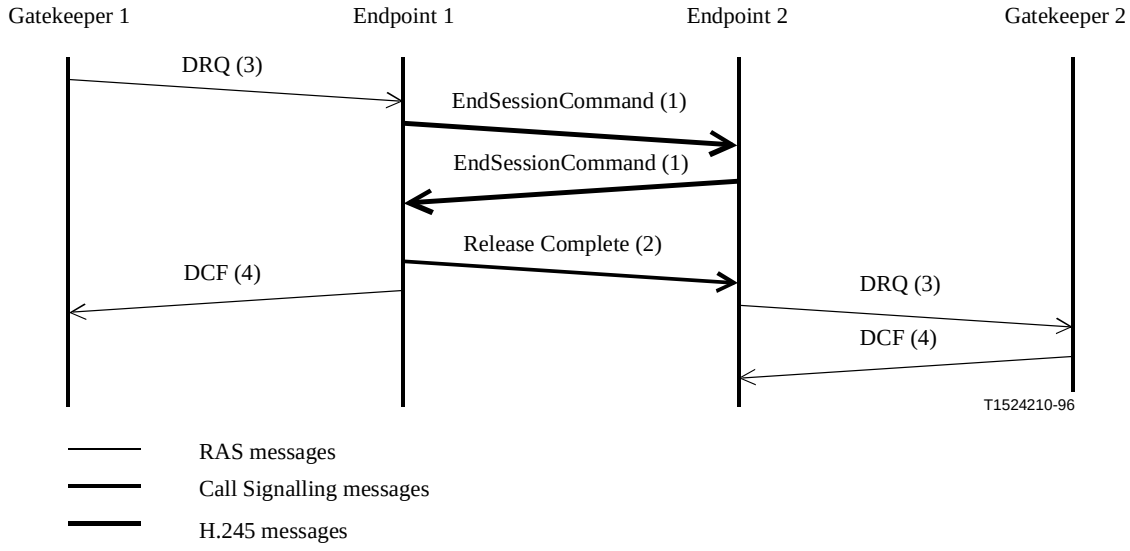
NOTE – Gatekeeper 1 and Gatekeeper 2 may be the same Gatekeeper.

Figure 32/H.323 – Endpoint initiated call clearing

8.5.3 Call clearing by Gatekeeper

The Gatekeeper may terminate call by sending a DRQ to an endpoint. See Figure 33. The endpoint shall immediately follow steps 1) through 6) from above and then reply to the Gatekeeper with DCF. The other endpoint, upon receiving **endSessionCommand,** shall follow the procedure described above. Figure 33 shows the direct call model; a similar procedure is followed for the Gatekeeper routed model.

If the conference is a multipoint conference, the Gatekeeper should send a DRQ to each endpoint in the conference, in order to close the entire conference.



NOTE – Gatekeeper 1 and Gatekeeper 2 may be the same Gatekeeper.

Figure 33/H.323 – Gatekeeper initiated call clearing

8.6 Protocol failure handling

The underlying reliable protocol of the H.245 Control Channel uses appropriate effort to deliver or receive data on the channel before reporting a protocol failure. Therefore, if a protocol failure is reported on the channel, the H.245 Control Channel, and all associated logical channels shall be closed. This shall be done following the procedures of Phase E, as if the other endpoint had issued the H.245 **endSessionCommand**. This includes transmission of the DRQ message to the Gatekeeper and termination of the Call Signalling Channel. In the case where the failure is detected by the MC in a multipoint conference, the MC shall send **terminalLeftConference** messages to the remaining terminals. It is up to the implementation whether or not to try to re-establish the call without user intervention. In any case, this would appear to the other endpoint (and the Gatekeeper) as a new call.

The Call Signalling Channel also uses an underlying reliable protocol. Depending on the routing of the Call Signalling Channel, either the Gatekeeper or an endpoint may detect the protocol failure. If the Gatekeeper detects the failure, it shall attempt to re-establish the Call Control Channel. This implies that the endpoint shall always have the ability to establish a channel on its Call Signalling Channel Transport Address. Failure of the Call Signalling channel shall not change the Q.931 call state. After re-establishment of the Call Signalling Channel, the Gatekeeper may send a Status message to request the call state of the endpoint to assure that they are in synchronization.

If the endpoint detects the failure, the endpoint may choose to terminate the call as described in Phase E, or it may attempt to re-establish the Call Signalling Channel as described above.

If, during a call, an endpoint wants to determine if the other endpoint is still functioning and connected, it may send the H.245 **roundTripDelayRequest**. Since H.245 Control Channel is carried on a reliable channel, this will result in a response from the other endpoint or an error from the transport interface. In the latter case, the procedures described above shall be used. An endpoint in a multipoint conference may use the same mechanism; however, it will learn only whether it still has a connection to the MC. Note that it is possible for an endpoint to have an error-free connection with the MC but still be receiving no audio or video from the rest of the terminals in the conference.

9 Interoperation with other terminal types

Interoperation with other terminals shall be accomplished through the Gateway. See 6.3.

9.1 Speech only terminals

Interoperation with speech only terminals (telephony) over the ISDN or GSTN can be provided by:

1) using a H.323-ISDN speech Gateway;
2) using a H.323-GSTN speech Gateway.

The Gateway should consider the following issues:

- Audio code conversion:
  - ISDN: If desired, since ISDN uses G.711.
  - GSTN: from analogue to G.711.
- Bit stream conversion:
  - ISDN: H.225.0 to/from unframed.
  - GSTN: generate H.225.0.
- Control conversion (generate H.245).
- Call Control Signalling conversion.
- DTMF tone conversion to/from H.245 **userInputIndication** message.

9.2 Visual telephone terminals over the ISDN (H.320)

Interoperation with visual telephone terminals over the ISDN (H.320) can be provided by:

– using a H.323-H.320 Gateway.

The Gateway should consider the following issues:

– Video format conversion. (If desired, H.261 is mandatory for both terminal types.)
– Audio code conversion. (If desired, G.711 is mandatory for both terminal types.)
– Data protocol conversion.
– Bit stream conversion. (H.225.0 to/from H.221.)
– Control conversion. (H.245 to/from H.242.)
– Call Control Signalling conversion.
– SBE Number conversion to/from H.245 **userInputIndication** message.

9.3 Visual telephone terminals over GSTN (H.324)

Interoperation with visual telephone terminals over the GSTN (H.324) can be provided by two methods:

1) using a H.323-H.324 Gateway;
2) using a H.323-H.320 Gateway, assuming that there exists an ISDN/GSTN Interworking Unit in the network.

The Gateway should consider the following issues:

– Video format conversion. (If desired, H.261 is mandatory for both terminal types.)
– Data protocol conversion.
– Audio code conversion. (G.711 is mandatory for H.323 terminal, G.723 is mandatory for H.324 terminal.)
– Bit stream conversion. (H.225.0 to/from H.223.)
– Call Control Signalling conversion.

9.4 Visual telephone terminals over mobile radio (H.324/M)

For further study.

9.5 Visual telephone terminals over ATM (H.321)

Interoperation with visual telephone terminals over ATM networks (H.321) can be provided by two methods:

1) using a H.323-H.321 Gateway;
2) using a H.323-H.320 Gateway, assuming that there exists an I.580 ISDN/ATM Interworking Unit in the network.

The Gateway should consider the following issues:

– Video format conversion. (If desired, H.261 is mandatory for both terminal types.)
– Data protocol conversion.
– Audio code conversion. (If desired, G.711 is mandatory for both terminal types.)
– Bit stream conversion. (H.225.0 to/from H.221.)
– Control conversion. (H.245 to/from H.242.)

– Call Control Signalling conversion.

9.6 Visual telephone terminals over guaranteed quality of service LANs (H.322)

Interoperation with visual telephone terminals over Guaranteed Quality of Service LANs (H.322) can be provided by:

– using a H.323-H.320 Gateway, assuming that there exists a GQOS LAN-ISDN Gateway in the network.

The Gateway should consider the following issues:

– Video format conversion. (If desired, H.261 is mandatory for both terminal types.)
– Data protocol conversion.
– Audio code conversion. (If desired, G.711 is mandatory for both terminal types.)
– Bit stream conversion. (H.225.0 to/from H.221.)
– Control conversion. (H.245 to/from H.242.)
– Call Control Signalling conversion.

9.7 Simultaneous voice and data terminals over GSTN (V.70)

Interoperation with Simultaneous Voice and Data Terminals over GSTN (V.70) can be provided by:

– using a H.323-V.70 Gateway.

The Gateway should consider the following issues:

– Audio code conversion. (G.711 to/from Annex A/G.729.)
– Data protocol conversion.
– Bit stream conversion. (H.225.0 to/from V.76/V.75.)
– Control conversion. (Both terminals use H.245.)
– Call Control Signalling conversion.

9.8 T.120 terminals on the LAN

An H.323 terminal that has T.120 capability should be capable of being configured as a T.120-only terminal which listens and transmits on the standard T.120 well-known TSAP Identifier. This will allow the T.120 capable H.323 terminal to participate in T.120-only conferences.

A T.120-only terminal on the LAN shall be able to participate in the T.120 portion of multipoint H.323 conferences. See 6.2.7.1.

10 Optional enhancements

10.1 Encryption

For further study.

11 Maintenance

11.1 Loopbacks for maintenance purposes

Some loopback functions are defined in Recommendation H.245 to allow verification of some functional aspects of the terminal, to ensure correct operation of the system and satisfactory quality of the service to the remote party.

The **systemLoop** request and **logicalChannelLoop** request shall not be used. The **mediaLoop** request is optional. An endpoint that receives the **maintenanceLoopOffCommand** shall turn off all loopbacks currently in effect.

For the purpose of loopbacks, two modes are defined:

a) Normal operation mode: No loopback. Indicated in **a)** of Figure 34. This shall be the default mode, and the mode entered when the **maintenanceLoopOffCommand** is received.

b) Media loop mode: Loopback of media stream at the analogue I/O interface. Upon receiving the **mediaLoop** request as defined in Recommendation H.245, loopback of the content of the selected logical channel shall be activated as close as possible to the analogue interface of the video/audio codec towards the video/audio codec, so that decoded and re-coded media content is looped, as indicated in **b)** of Figure 34. This loopback is optional. It should be used only when a single logical channel containing the same media type is opened in each direction. Operation when multiple channels are opened in the return direction is undefined.

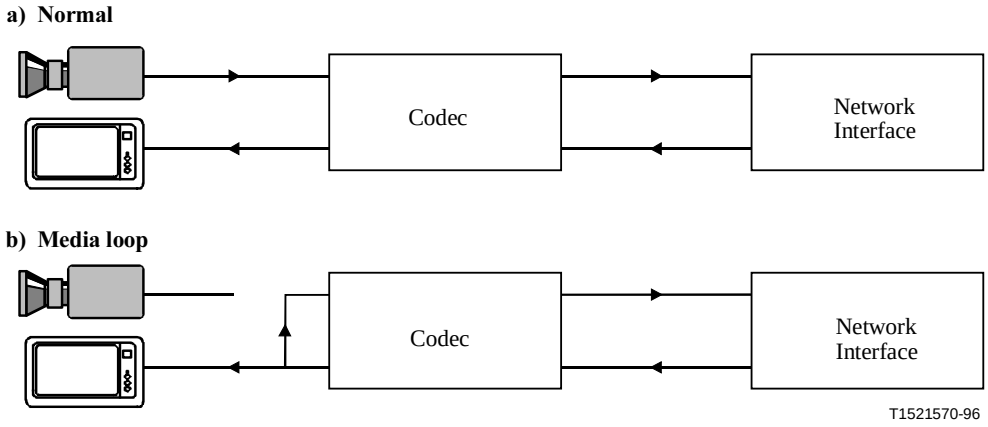


Figure 34/H.323 – Loop back

A Gateway to H.324 and H.310, which receives an H.245 **systemLoop** request, H.245 **logicalChannelLoop** request, or a Gateway to H.320, which receives an H.230 Dig-Loop command from an SCN endpoint may perform the appropriate loopback function within the Gateway. The Gateway shall not pass these requests to the LAN endpoint. A Gateway to H.324/H.310, receiving H.245 **mediaLoop** from an SCN endpoint shall pass the request to the LAN endpoint. A Gateway to H.320, receiving H.230 Vid-loop or Au-loop command from an SCN endpoint shall convert it to the appropriate H.245 **mediaLoop** request and send it to the LAN endpoint.

A Gateway H.320, which receives an H.245 **mediaLoop** request from a LAN endpoint shall convert it to the appropriate H.230 Vid-loop or Au-loop command and send it to the SCN endpoint.

A Gateway to H.324 and H.310, may send an H.245 **systemLoop** request or H.245 **logicalChannelLoop** request to the SCN endpoint. A Gateway to H.320 may send an H.230 Dig-Loop command to the SCN endpoint. If a LAN endpoint is in a call to the SCN endpoint, the

audio and video sent to the LAN endpoint may be the looped back audio or video, pre-recorded audio or video message indicating the loopback condition, or no audio or video.

11.2 Monitoring methods

All terminals shall support the Information Request/Information Request Response (IRQ/IRR) message of Recommendation H.225.0. The Information Request Response message contains the TSAP Identifier of all channels currently active on the call, including T.120 and H.245 control, as well as audio and video. This information can be used by third party maintenance devices to monitor H.323 conferences to verify system operation.

ANNEX A

H.245 messages used by H.323 endpoints

The following rules apply to the use of H.245 messages by H.323 endpoints:

- An endpoint shall not malfunction or otherwise be adversely affected by receiving H.245 messages that it does not recognize. An endpoint receiving an unrecognized request, response, or command shall return "function not supported". (This is not required for indications.)
- The following abbreviations are used in Tables A.1 to A.11:

  M Mandatory

  O Optional

  F Forbidden to transmit
- A message marked as mandatory for the receiving endpoint indicates that the endpoint shall accept the message and take the appropriate action. A message marked as mandatory for the transmitting endpoint indicates that the endpoint shall generate the message under the appropriate circumstances.

Table A.1/H.323 – Master-slave determination messages

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Determination	M	M
Determination Acknowledge	M	M
Determination Reject	M	M
Determination Release	M	M

Table A.2/H.323 – Terminal capability messages

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Capability Set	M	M
Capability Set Acknowledge	M	M
Capability Set Reject	M	M
Capability Set Release	M	M

Table A.3/H.323 – Logical channel signalling messages

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Open Logical Channel	M	M
Open Logical Channel Acknowledge	M	M
Open Logical Channel Reject	M	M
Open Logical Channel Confirm	M	M
Close Logical Channel	M	M
Close Logical Channel Acknowledge	M	M
Request Channel Close	M	O
Request Channel Close Acknowledge	O	O
Request Channel Close Reject	O	M
Request Channel Close Release	O	M

Table A.4/H.323 – Multiplex table signalling messages

Message	Status
Multiplex Entry Send	F
Multiplex Entry Send Acknowledge	F
Multiplex Entry Send Reject	F
Multiplex Entry Send Release	F

Table A.5/H.323 – Request multiplex table signalling messages

Message	Status
Request Multiplex Entry	F
Request Multiplex Entry Acknowledge	F
Request Multiplex Entry Reject	F
Request Multiplex Entry Release	F

Superseded by a more recent version

Table A.6/H.323 – Request mode messages

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Request Mode	M	O
Request Mode Acknowledge	M	O
Request Mode Reject	O	M
Request Mode Release	O	M

Table A.7/H.323 – Round trip delay messages

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Round Trip Delay Request	M	O
Round Trip Delay Response	O	M

Table A.8/H.323 – Maintenance loop messages

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Maintenance Loop Request		
System Loop	F	F
Media Loop	O (Note)	O (Note)
Logical Channel Loop	F	F
Maintenance Loop Acknowledge	O	O
Maintenance Loop Reject	O	M
Maintenance Loop Command Off	M	O
NOTE – Mandatory in Gateways.		

Superseded by a more recent version

Table A.9/H.323 – Commands

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Send Terminal Capability Set	M	M
Encryption	O	O
Flow Control	M	O
End Session	M	M
Miscellaneous Commands		
Equalize Delay	O	O
Zero Delay	O	O
Multipoint Mode Command	M	O
Cancel Multipoint Mode Command	M	O
Video Freeze Picture	M	O
Video Fast Update Picture	M	O
Video Fast Update GOB	M	O
Video Fast Update MB	M	O
Video Temporal Spatial Trade Off	O	O
Video Send Sync Every GOB	O	O
Video Send Sync Every GOB Cancel	O	O
MCLocationIndication	M	O
Terminal ID Request	O	O
Terminal List Request	O	O
broadcast Me	O	O
cancel Broadcast Me	O	O
Make Terminal Broadcaster	O	O
Send This Source	O	O
Cancel Send This Source	O	O
Drop Terminal	O	O
Make Me Chair	O	O
Cancel Make Me Chair	O	O
Drop Conference	O	O
Enter H.243 Password	O	O
Enter H.243 Terminal Id	O	O
Enter H.243 Conference ID	O	O
Request Terminal ID	O	O
Terminal ID Response	O	O
Terminal List Response	O	O
Video Command Reject	O	O
Make Me Chair Response	O	O

Table A.10/H.323 – Conference mode commands

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Communication Mode Command	M	O
Communication Mode Request	O	O
Communication Mode Response	O	O

Table A.11/H.323 – Indications

Message	Receiving Endpoint Status	Transmitting Endpoint Status
Function Not Supported	M	M
Miscellaneous Indication		
Logical Channel Active	O	O
Logical Channel Inactive	O	O
Multipoint Conference	M	O
Cancel Multipoint Conference	M	O
Multipoint Zero Comm	O	O
Cancel Multipoint Zero Comm	O	O
Multipoint Secondary Status	O	O
Cancel Multipoint Secondary Status	O	O
Video Indicate Ready to Activate	O	O
Video Temporal Spatial Trade Off	O	O
SBE Number	O	O
Terminal Number Assign	M	O
Terminal Joined Conference	O	O
Terminal Left Conference	O	O
Seen By At Least One Other	O	O
Cancel Seen By At Least One Other	O	O
Seen By All	O	O
Cancel Seen By All	O	O
Terminal You Are Seeing	O	O
Request For Floor	O	O
Jitter Indication	O	O
H.223 Skew Indication	F	F
H2250MaximumSkewIndication	O	M
New ATM Virtual Channel Indication	F	F
User Input	M (for 0-9, * and #)	M (for 0-9, * and #)

Non-standard commands, requests, etc. are allowed.

APPENDIX I

Processing of Q.931 messages in Gateways

The Gateway shall terminate the Q.931 Call Signalling Channel between an H.323 endpoint and the Gateway on one hand and the call signalling channel (if any) between the Gateway and the SCN endpoint on the other. The following applies only if the SCN side supports a call signalling protocol such as Q.931 or Q.2931.

The Gateway shall conform to the call signalling procedures recommended for the SCN side independent from the LAN side. The Gateway shall conform to the call signalling procedures of this Recommendation for the LAN side independent from the SCN side.

In addition, call signalling messages received from one side (LAN/SCN) may require forwarding to the other side (SCN/LAN). Some forwarded messages may contain information elements or parts of information elements which are unmodified or uninterpreted by the Gateway. Other forwarded messages may contain information elements or parts of information elements may be added or removed by the Gateway, as needed.

In the following, an overview of the actions to be taken by the gateway in response to Q.931 messages and the information elements, is provided. Messages and information elements that are forbidden in Recommendation H.225.0 are not considered.

Q.931 messages originating on the H.323 side:

- A SETUP message side shall lead to initiation of call set-up procedure for the SCN side.
- A RELEASE COMPLETE shall lead to initiation of the call disconnect as defined for the SCN side.
- A CALL PROCEEDING message shall be forwarded to the SCN side. This shall not be done if a CALL PROCEEDING has been sent before to the SCN in compliance to the respective SCN specification (Q.931 in the ISDN case).
- A CONNECT message shall be forwarded to the SCN side upon receipt from an H.323 endpoint.
- A CONNECT ACKNOWLEDGE message shall be forwarded to the SCN side upon receipt. This shall not be done if CONNECT ACKNOWLEDGE has been sent before to the SCN in compliance to the respective SCN specification. If the Gateway has sent a CONNECT to an H.323 endpoint and has not received the corresponding CONNECT ACKNOWLEDGE from the H.323 endpoint within the time frame required for successful completion of the call, it shall generate the CONNECT ACKNOWLEDGE to the SCN side as appropriate to comply to the SCN call set-up procedures.
- Messages for supplementary services (FACILITY, HOLD, HOLD ACKNOWLEDGE, HOLD REJECT, RETRIEVE, RETRIEVE ACKNOWLEDGE, RETRIEVE REJECT and the INFORMATION messages) that are not processed by the Gateway, shall be forwarded to the SCN side.
- All messages forbidden to be originated from an H.323 endpoint shall be generated by the Gateway autonomously as required by the SCN protocol.

The information elements of the respective messages are to be converted as follows:

- The contents of connection specific information elements (such as Call Reference Value) shall be adapted as required by the SCN protocol.

- Information elements that are not in use on the H.323 side shall be generated by the Gateway as required by the SCN protocol.
- Translation of other information elements shall be done as required by the SCN protocols and procedures. Where interoperability is not an issue, conversion is left to the discretion of the manufacturer.
- Only the user-data part of the user-user information element shall be forwarded to the SCN side. It shall be re-encoded following Figure 4-36/Q.931 and Table 4-26/Q.931.

All Q.931 messages originating on the SCN side are forwarded to the H.323 endpoint without modification except for the following:

- Messages forbidden by Recommendation H.225.0 shall not be passed to the H.323 side.
- The call reference value is mapped to the appropriate value for the H.323 side.
- The user data field is copied into the corresponding ASN.1 user-user information element structure.
- The user-user information element structure shall be generated according to the specification in Recommendation H.225.0.

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	**Audiovisual and multimedia systems**
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communication
Series Z	Programming languages

Printed in Switzerland, Geneva - 1996

H

H.323 An *ITU-TSS* standard that defies standards for real-time Internet telephony as well as audio and video conferencing via the Internet. Applications that conform to H.323 can work with other H.323-compliant programs, even if they are made by different companies.

hack **1.** A clever and original rearrangement of the existing system or network resources that results, as if by magic, in a stunning improvement in system performance (or an equally stunning prank). A hacker is one who uses computers to perform hacks and is not necessarily a computer criminal. See *cracker, hacker ethic, phreaking.* **2.** A "quick and dirty" job that produces results, but without following any logical or orderly procedure.

hacker **1.** A computer enthusiast who enjoys learning everything about a computer system or network and through clever programming, pushing the system to its highest possible level of performance. **2.** In the press, synonymous with *cracker.* **3.** An adept programmer.

hacker ethic A set of moral principles that were common to the first-generation hacker community (roughly 1965—1982), described by Steven Levy in *Hackers* (1984). According to the hacker ethic, all technical information should, in principle, be freely available to all. Therefore, gaining entry to a system to explore data and increase knowledge is never unethical. However, destroying, altering, or moving data in such a way that could cause injury or expense to others is always unethical. In increasingly more states, unauthorized computer access is against the law. See *cracker, cyberpunk, cyberspace, computer ethics, hack,* and *phreaking.*

half-duplex An asynchronous communications protocol in which the communications channel can handle only one signal at a time. The two stations alternate their transmissions. Synonymous with *local echo.* See *communications protocol, echoplex,* and *full duplex.*

half-height drive A disk drive half the size of the three-inch-high drives in the original IBM Personal Computer. Half-height drive bays and drives are standard in today's PCs.

output simulation *n.* A feature of color management applications in which a computer display is calibrated to help predict the results of printing a graphics file on a specific device. *Also called:* soft proofing.

output stream *n.* A flow of information that leaves a computer system and is associated with a particular task or destination. In programming, an output stream can be a series of characters sent from the computer's memory to a display or to a disk file. *Compare* input stream.

outsourcing *n.* The assignment of tasks to independent contractors, such as individual consultants or service bureaus. Tasks such as data entry and programming are often performed via outsourcing.

OverDrive *n.* A type of microprocessor from Intel designed to replace a computer's existing i486SX or i486DX microprocessor. The OverDrive is functionally identical to Intel's i486DX2 microprocessor, but it is an end-user product, whereas the i486DX2 is sold only to computer manufacturers who build it into their own systems. Upgrading a system with an OverDrive processor differs from system to system, and some systems might not be able to support an OverDrive processor. *See also* i486DX, i486SL, i486SX, microprocessor. *Compare* i486DX2.

overflow *n.* **1.** Generally, the condition that occurs when data resulting from input or processing requires more bits than have been provided in hardware or software to store the data. Examples of overflow include a floating-point operation whose result is too large for the number of bits allowed for the exponent, a string that exceeds the bounds of the array allocated for it, and an integer operation whose result contains too many bits for the register into which it is to be stored. *See also* overflow error. *Compare* underflow. **2.** The part of a data item that cannot be stored because the data exceeds the capacity of the available data structure.

overflow error *n.* An error that arises when a number, often the result of an arithmetic operation, is too large to be contained in the data structure that a program provides for it.

overhead *n.* Work or information that provides support—possibly critical support—for a computing process but is not an intrinsic part of the operation or data. Overhead often adds to processing time but is generally necessary.

overlaid windows *n.* *See* cascading windows.

overlapped communication operation *n.* The performance of two distinct communication operations simultaneously; for example, a simultaneous read/write operation. Windows CE does not support overlapped communication operation, but it does support multiple read/writes pending on a device.

overlay[1] *n.* **1.** A section of a program designed to reside on a designated storage device, such as a disk, and to be loaded into memory when needed, usually overwriting one or more overlays already in memory. Use of overlays allows large programs to fit into a limited amount of memory, but at the cost of speed. **2.** A printed form positioned over a screen, tablet, or keyboard for identification of particular features. *See also* keyboard template.

overlay[2] *vb.* **1.** In computer graphics, to superimpose one graphic image over another. **2.** In video, to superimpose a graphic image generated on a computer over video signals, either live or recorded.

overprint *vb.* The process of printing an element of one color over one of another color without removing, or knocking out, the material underneath. *Compare* knockout (definition 1).

override *vb.* To prevent something from happening in a program or in an operating system or to initiate another response. For example, a user can often override and thus abort a lengthy sorting procedure in a database program by pressing the Escape key.

overrun *n.* In information transfer, an error that occurs when a device receiving data cannot handle or make use of the information as rapidly as it arrives. *See also* input/output-bound.

overscan *n.* The part of a video signal sent to a raster display that controls the area outside the rectangle containing visual information. The overscan area is sometimes colored to form a border around the screen.

overshoot *n.* The phenomenon in which a system suffers from a time delay in responding to input and continues to change state even after it has reached the desired state. This situation requires that correcting input be made so that the system reaches the desired state. For example, the arm carrying the heads in a hard disk drive might move slightly past the desired track before it stops, requiring another signal to pull it back.

overstrike *vb.* To type or print one character directly over another so that the two occupy the same space on the page or screen.